Oil recovery in porous media using emulsion flooding

by
Aleksey Baldygin

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Mechanical Engineering
University of Alberta

Abstract

Aiming to pursue studying enhanced oil recovery processes with an *ex-situ* produced emulsion as flooding agent, an improved core flooding experimental system was designed, constructed and commissioned. The developed system allows the use of not only emulsion, but also other flooding agents, including surfactants and polymers. The components of the experimental apparatus, including a biaxial core holder, a two-dimensional core holder, and effluent analysis system were re-designed to improve flow distribution, the packing process and reduce post-experimental analysis. To secure the conditions for experiments, custom software for system control and monitoring, and logging of all system parameters was developed. The limits and ways to improve the developed system were identified. An unconsolidated silica sand pack was chosen to represent the porous media on all experimental tests.

The initial experimental results with emulsion flooding explored the capability of the new apparatus and confirmed the efficiency of emulsion as a flooding agent during direct emulsion flooding or water flooding followed by emulsion flooding. Oil recovery was enhanced by 15 % of original oil in place in case of emulsion flooding followed by water flooding. From experimental results, it was hypothesised that emulsion moves through the sand pack with a piston-like flood front and

that an alternative technique to improve efficiency of emulsion flooding can be developed. This technique should also allow to reduce amount of emulsion used per percentage of oil recovery.

A flooding technique was proposed and established to improve the efficiency of emulsion flooding in reservoirs and reduce costs of flooding. The technique was named water-alternate-emulsion (WAE). Three different flooding ratios have been tested using a biaxial core holder. The experimental analysis showed that for 5:1 WAE flooding, the ultimate oil recovery could be up to 82 % at 0.3 PV of total emulsion injection. The analysis of physical properties of the effluent, such as viscosity, density and surface tension, revealed that alternate flooding does not affect on physical properties of liquids involved in flooding. Effluent analysis and examination of extracted sand packs after experiment completion reported that emulsion undergoes separation inside porous media during alternative flooding and moves with a post-like flood front during continuous injection. As a results, it blocks high-permeability zones and surfactant present in emulsion interacts with *in-situ* liquids. Pressure measurements identified that emulsion stimulates sand packs with internal pressure rise prior to water flooding.

Experimental flow visualization and pressure mapping analysis of emulsion flooding with horizontal and vertical wells was performed to provide additional understanding of internal processes. It was shown experimentally, that efficiency of emulsion flooding strongly depends on well configuration. The horizontal injection - horizontal production (HI-HP) configuration for injection and production well, based on data from effluent and image analysis, was found to be the most efficient for emulsion flooding. The flood front movement during emulsion flood-

ing process was estimated using code developed for image analysis. Experimental data presented in this research project suggests that stable emulsion flood front and stable pressure rise guarantees enhanced oil recovery during emulsion injection. In addition, oil drops present in an emulsion were observed with a microscope and it was hypothesised that emulsion has bimodal structure. Later it was confirmed with the membrane-base particle analysers.

As a results of this research study, emulsion flooding process was investigated experimentally. It was shown that emulsions can be used effectively to recover oil using different flooding techniques. Oil-in-water emulsion blocked highly permeable zones and redirecting following water flooding. Experimental results from multi-dimensional studies suggested that it is important to consider well configuration and physical parameters, such as pressure data, in a field trials. Collected results for the drop size distribution now can be considered in a future fundamental works for emulsion flooding process.

Preface

This thesis is original work by Aleksey Baldygin. Chapter 3 and 4 of this thesis have been published as A. Baldygin, D. S. Nobes, and S. K. Mitra, "A new laboratory core flooding experimental system", *Industrial & Engineering Chemistry Research*, DOI: 10.1021/ie501866e, 2014 and as A. Baldygin, D. S. Nobes, and S. K. Mitra, "Water-Alternate-Emulsion (WAE): A new technique for enhanced oil recovery",*Journal of Petroleum Science and Engineering*, DOI:10.1016/j.petrol.2014.06.021, 2014, respectively. Chapter 5 has been prepared for submissions as publication as A. Baldygin, D. S. Nobes, and S. K. Mitra, "Experimental pressure mapping and flow visualization of emulsion flooding with horizontal and vertical wells", *Journal of Petroleum Science and Engineering*, 2014. Appendix A-2 has been published as part of the conference paper as A. Baldygin, D. S. Nobes, and S. K. Mitra, "Oil recovery from porous media using emulsion", *Proceedings of the ASME 2014 International Mechanical Engineering Congress & Exposition*, 2014. The initial concept of the experimental system has been presented during the biennial CSME International Congress: Symposium on Advanced Energy Systems (CSME'12 - Symposium on AES) as A. Baldygin, D. S. Nobes, and S. K. Mitra, "Novel Technique for Core Flooding Experiments", 2012. The software with graphical user interface was developed myself and used to assist experiments in the current re-

search study. The design of core holders with appropriate drawing package was also developed myself. In addition, I was responsible for data collection and analysis as well as for manuscript preparation. Dr. David S. Nobes and Dr. Sushanta K. Mitra were the supervisory authors and contributed to manuscript edits. Emulsion used in experiments was provided by Quadrise Canada Inc..

Acknowledgements

I would like to thank my supervisor, Dr. David S. Nobes for believing in me in 2011 and giving me an opportunity to be part of PhD program at the Mechanical Engineering department. I am also thankful for the productive discussions that we had along my program. He gave me a passion to have attention on small details and provided continuous support during my program. I would also like to thank my co-supervisor, Dr. Sushanta K. Mitra for the time that he spend with me over weekends and helped me to improve my writing and thinking skills. His guidance and support over the last three years provided me an opportunity to do up-to-date level research.

My thanks and appreciation to Roger Marchand, Andrew Campbell, Rick Bubenko and other shop technicians who have provided me technical and engineering support whenever I needed it. I would also thank Rick Conrad for his expertise when I was setting up experimental set-up for the first time.

I would also like to thank my friends and colleagues: Anil Stephen, Prashant Waghmare, Orest Shardt, Lalit Pant, Arnab Guha, Sheng Tian, Amrit Bhinder, Reza Sabbagh and many others with whom I spent last three years. They have supported my research and my recreation time whenever I needed.

In addition, I would like to acknowledge Dr. Patrick Brunelle from Quadrise Canada Inc. for his valuable inputs and providing emulsion for experiments. Financial assistance from NSERC CRD (Grant No. CRDPJ 399500-10) is acknowledged here.

A sincere thank you to my wife Alexandra Komrakova and my little daughter Olivia Baldygin for their support, love and patience. Finally, a greatest thank you to my parents for providing me support during my whole live and believing that one day, I could be a PhD.

Contents

List of Tables

List of Figures

Nomenclature

A	Cross section
d	Diameter in 1D cell
h	Sand pack height
H_{in}	Hydraulic head inlet
H_{out}	Hydraulic head outlet
K	Hydraulic conductivity
k, k_a	Absolute permeability
k_e	Effective permeability
k_r	Relative permeability
L	Length of cell
M	Mobility
P_c	Capillary pressure
P_{crit}	Critical pressure

p_{in} Water pressure at the inlet

P_m Mean absolute pressure

p_{out} Water pressure at the outlet

Q Discharge from porous media

q Darcy flux

Q_{cur} Current volume flow rate

R Droplet radius

r Pore throat radius

r^*, R^* Radii

S Fluid saturation

t_{step} Single time step

u_t Darcy velocity in linear displacement

U_{in} Interstitial velocity

U_o Superficial (or empty tube) velocity

V Bulk volume

v Velocity of the displacing fluid

V_d Dead volume

V_{eff} Effective pore volume

V_p, V_{pv} Pore volume

V_s Grain volume

V_v Void space

$V_{j,rec}$ Fluid volume for a given phase

V_{kv} Volume of water in water tank

V_{rem}, V_{water_rec} Remaining volume of water in water tank

w Sand pack width

z_{in} Porous media elevation at the inlet

z_{out} Porous media elevation at the outlet

Greek Letters

$\dot{\gamma}$ Share rate

η, η_1 Viscosity of displacing fluid

γ, γ_{12} Interfacial tension

λ Mobility ratio

Θ Contact angle

μ Dynamic viscosity

ϕ Porosity

ρ_g Grain density

ρ_{record} Recorded fluid density

ρ_{ref} Reference fluid density

Subscript

e Emulsion

g Gas

i Row number (used in Chapter 5)

j Oil or water, based on readings; column number (used in Chapter 5)

n Number for Run/Experiment

o Oil

o/w Oil and water

r Relative

w Water

Dimensionless numbers

$We = \frac{\eta_1 \dot{\gamma} R}{\gamma_{12}}$ Weber number

Abbreviations

1D One dimensional

2D Two dimensional

CAG Chemical-augmented WAG

CSS Cyclic steam simulation

DAQ Data acquisition

DG Double gap

DSA Drop shape analyzer

DSD Droplet size distribution

DSLR Digital single-lens reflex

EOR Enhanced oil recovery

GUI Graphical user interface

HI Horizontal injection

HP Horizontal production

ID Internal diameter

IOR Improved oil recovery

LDS Liquid dosage system

NDT Nondestructive testing

O/W Oil-in-water

OBIP Original bitumen in placeG

OD Outer diameter

OOIP Original oil in place

PA Piston accumulator

PBS Phosphate buffered saline

PC Personal computer

PV Pore volume

ROC Reservoir-on-a-Chip

SAGD Steam assisted gravity drainage

SMTP Simple mail transfer protocol

sRGB Standard Red Green Blue color space

VI Vertical injection

VP Vertical production

W/O Water-in-oil

WAE Water-alternate-emulsion

WAG Water-alternate-gas

Chapter 1

Introduction

1.1 Motivation

For at least the next decade, fossil fuels, such as gas and oil will remain the dominant energy source (Dresselhaus and Thomas, 2001; Omrpublic.iea.org, 2014). Gas and oil are typically trapped inside the naturally formed pores in a geological formation (referred to porous media). The process of oil recovery can be divided into three stages. The first stage is called primary recovery and the efficiency of oil recovery at this stage is controlled by the geological formation. The primary stage typically allows recovery of 5-15 % of original oil in place (OOIP) (Tzimas et al., 2005). The second stage of oil recovery which is usually a form of water flooding, provides an additional 30-35 % (Green and Willhite, 1998). The final stage is tertiary recovery which has potential for high oil recovery ($\approx$ 80 - 90 %). The last two stages require the application of pressure to the system.

Different research groups are working on the improvement of current tertiary methods that include thermal recovery, surfactant/polymer flooding, emulsion flooding, and CO_2 injection (Liu et al., 2007; Asghari and Nakutnyy, 2008; Shen et al., 2009; Alvarado and Manrique, 2010; Hart, 2013; Hein et al., 2013). These methods

allowed to achieve record levels of oil production (92 million barrels/day (mb/d)) (Omrpublic.iea.org, 2014). Emulsion flooding is one of the improved oil recovery (IOR) and enhanced oil recovery (EOR) techniques used for oil recovery. Emulsion (oil-in-water or water-in-oil) produced *ex-situ* as a fuel source for industrial and commercial, utility, marine engines or boilers (www.altpetrol.com, 2014) can be used either during the secondary stage coupled with water flooding or as independent flooding during the tertiary stage. Process of residual oil emulsification has established itself as an efficient method for enhanced oil recovery (Speight, 2006). It can be used for recovering heavy crude oil with gravity from 5° to 20° API (Speight, 2006).

Heavy oil reserves in Canada exceed a trillion barrels (Adams, 1982) and it was estimated that only 8 - 9 % of the OOIP were recoverable (Adams, 1982) with available technologies on the market in 1982. Improved technologies allowed increased production up to 15 - 20% (Hein et al., 2013; Larter and Head, 2014). Steam-assisted gravity drainage (SAGD) and cyclic steam stimulation are used to recover heavy oil from thin reservoirs (Hart, 2013; Arhuoma et al., 2009). However, most heavy oil is available in the northern territories of Canada. In the case of shallow oil, thermal recovery can damage the environment (Qiu and Mamora, 2010). There is a possibility for high-pressure steam escaping to the surface with resulted damage to the ambient environment. In opposite, emulsion flooding technologies can already be used in proximity to the permafrost without possibility damaging a surface with a steam (Qiu and Mamora, 2010). However, there is a potential effect on nature from possible spills and usage of fresh water during oil production. These effects must be considered before attempting field trials.

Emulsions are mixtures of oil and water which can be obtained through the application of energy through mixing or friction (Schramm, 2006, p. 138). The essential parameters of emulsions are its type, *i.e.* water-in-oil (W/O) or oil-in-water (O/W), droplet size distribution (DSD), rheology (Newtonian or non-Newtonian fluid), and apparent density (ρ). It is of great importance to obtain these parameters during the production of the emulsion and keep them constant during its lifetime since they influence the efficiency of emulsion flooding (Wasan et al., 1978). The stability of the emulsion can be controlled by surfactant or by variation of DSD (Schramm, 2006, p. 138-139). Emulsion, which was used in this study, is a product of Quadrise Canada Inc. under brand name E^2EOR. Their technology allows to tailor parameters for any particular application, such as viscosity, hydrocarbon content, the drop size distribution, chemistry and stability.

Emulsions can be used to decrease mobility of the injected fluids (extrusive liquid) or to block highly permeably zones (McAuliffe, 1973; Mandal et al., 2010a). Core flooding experiments can be used to provide information on the emulsion flooding process (Samanta et al., 2012; Moradi et al., 2014). The surfactant added to the emulsion reduces the interfacial tension of the produced fluid when it interacts within the porous media with oil (Kumar et al., 2010). Oil will be recovered first from small pores in water-wet reservoirs where these small pores are connected to each other and will create a highly permeable zone through which water can flow (Schramm, 2006, p. 268). This can be compared to the emulsion produced from crude oil which consists of the high viscous drops. These drops require higher critical pressure to squeeze and move through and out from the pore head. With known size of the pores, a designed emulsion can be produced with known drop

size distribution which is sized to block zones of high permeability and to resist the pressure of the injected fluid. Hence, injected liquids are forced to flow through the low permeable zone and recover additional oil.

Decker and Flock (1988) carried out experiments and showed that emulsions are efficient in emulsion blockage mechanism. Also, emulsions have viscosities higher compared to water and allow higher capillary numbers to be generated. Higher capillary numbers describe the presence of higher viscous forces in pores that allow mobilization of the oil drops in the porous media (Moradi et al., 2014). Mandal et al. (2010a) concluded that emulsion flooding is more efficient as a displace fluid compared to traditional water flooding because of improved mobility ratio as a result of increase in viscosity of the displacing fluid (emulsion). However, Cobos et al. (2009) pointed out that understanding of this process requires further pore-scale studies.

To move from pore-scale to macro-scale to investigate and understand how the emulsion phase interacts within the porous structures of the reservoir and the efficiency of this process, laboratory core flooding experiments are required. Laboratory core flooding experiments include investigation of reservoir parameters. Every newly introduced method for oil recovery requires testing experiments with different flow patterns, well configurations and field parameters. For instance, one dimensional (1D) core holders allow the study of linear flow patterns and obtain initial parameters, such as porosity, permeability, efficiency for oil recovery and set effective flow rates and can provide initial understanding for oil recovery mechanism during emulsion flooding (Hadia et al., 2007a). Mechanisms involved in emulsion flooding, the increase of the mobility of injected fluids or the blockage

of highly permeably zones, can be investigated with 1D core flooding experiments. Based on parameters obtained during 1D studies, such as pressure drop, the amount of oil recovered after induction emulsion into the system, it would be possible to investigate the mechanism of the emulsion for EOR.

Understanding of emulsion flooding process at the tertiary stage, its interaction with oil saturated porous media in multi-dimensional experimental studies has also been considered as another part of the research. A two dimensional (2D) core holder allow investigation of how the front of the flooding agent spreads through the porous media. A 2D core holder represents a sectional cut of a real reservoir. It is thin enough to assume that the flow is distributed in two directions only and there is no influence (flow) in the third direction. A 2D core holder highlights the effects of well configurations (Santosh et al., 2007, 2008) and strata (Chaudhari et al., 2011a) providing data on emulsion distribution in two direction and its interaction with the porous media. Based on obtained data, the type of emulsion can be specified. There is a significant number of studies showing how to increase oil recovery during water flooding by configuring wells in appropriate orientations (Chaudhari et al., 2011a; Bagci and Gumrah, 1992; Hadia et al., 2007b). Consequently, it is essential to study the dependency of wells configuration during emulsion flooding to determine their effect on the efficiency of recovery. Taking pictures from one side of the 2D visualization cell provide a map for oil, water and emulsion saturations and velocity of emulsion flood front. Another side of the core holder can be equipped with pressure sensors to provide pressure mapping analysis for the *in-situ* processes.

Core flooding experiments with emulsion flooding experiments have been performed by different research groups (Qiu and Mamora, 2010; D'Elia-S and Ferrer-

G, 1973; Walstra, 1993; Mandal et al., 2010b; Guillen et al., 2012; Kumar et al., 2012). A dominant part of these research studies have focused on ultimate oil recovery with an emulsion formed *in-situ* (D'Elia-S and Ferrer-G, 1973; Walstra, 1993; Kumar et al., 2012). Flooding agents (e.g., surfactant (Samanta et al., 2012; Kumar et al., 2012; Bryan and Kantzas, 2009; Krumrine and J.S., 1983)) present in injected fluids emulsify trapped residual oil and form *in-situ* emulsion. Only a limited number of studies are available for *ex-situ* produced emulsion (Moradi et al., 2014; Mandal et al., 2010b; Guillen et al., 2012; Qiu, 2010; Nogueira et al., 2013). Experimental setups for both types of emulsion followed a traditional design for core flooding studies.

During emulsion flooding experiments, multi-phase flows are present in the system and the experimental setup design has a significant influence from this. The amounts of injected and produced liquids have to be analyzed, as well as the pressure along the system has to be recorded. Emulsion has physical properties close to oil and water and conventional techniques for real-time effluent analysis, such as segregation method (Qiu and Mamora, 2010), use of visual cells or use of an ultra-sound separators (Boye et al., 2008) are not feasible. This leads to complex post-experimental analysis, such as Dean-Stark analysis, mass balance calculations with liquid-liquid separation to obtain amount for each phase collected from the system. Qiu and Mamora (2010) recently used these calculations and showed that around an additional 20 % of OOIP was recovered during emulsion flooding. Similar results were obtained by Mandal et al. (2010a). They, also, showed that additional 20 % of OOIP was recovered during emulsion flooding. The obtained numbers highlight the potential for emulsion flooding; however, the slow and com-

plicated post-experimental analysis slows down the progress of these types of studies. Thus, it is necessary to develop a system with on-line volume measurements of each phase.

During the past twenty years, control and measurement technologies have stepped forward and brought a wide range of new devices to the market. Their usage allows the development of a system to provide an extended range of measured parameters and improve experimental control. This allows the development of a system for core flooding studies. In order to avoid routine post-experimental procedures it may be possible to develop analysis equipment for the downstream effluent. This equipment would analyze the flow and coupled with data from upstream and downstream mass flow meters gives the values for recent drainage. Online information would potentially lead to the exploration of new regimes of three-phase relative permeability curves relevant to a particular EOR technique. Developing experimental setup to study enhance oil recovery considers as a part of the study, to provide facility for understanding emulsions' potential in oil recovery process.

1.2 Thesis objective

The main objective of this study is to contribute to the global understanding of emulsion flooding by studying the process of enhance oil recovery with emulsion as flooding agent. The aim is to investigate the technical feasibility of using emulsion as recovery agent and to develop efficient method for using in a field. Present study does not include an economical analysis of the emulsion flooding process and mainly focuses on efficiency. It is worth emphasizing that the results of this research can be used to develop theory for emulsion flooding using *ex-situ* produced

emulsion and to guide field engineers during field trials.

The reservoir is a region of porous medium with pores and capillaries. In order to efficiency recover the oil, it is necessary to understand what traps oil within the reservoir. Conditions of the reservoir after and during the first two recovery stages impact the parameters of tertiary oil stage. It is important to know these conditions and the physics behind them. The second chapter of this thesis focuses on understanding the conditions and the physics of emulsion flooding in porous media and provides required background.

A literature review of existing experimental systems and a review of current technologies is described in details in Chapter 3. Based on obtained results, a system has been designed, procured, constructed, commissioned, and initial experiments with emulsion flooding were carried out, details are available in Chapter 3. Also, as a result of performing routine experiments, the method for effluent analysis based on the data from a mass flow meter has been developed and tested. In addition, the traditional design of the biaxial core holder has been modified to reduce internal dead volumes during flooding and improve flow stability.

A hypothesis of alternative flooding technique efficiency is proposed and tested and compared to the the previously accomplished traditional floodings in Chapter 3. As a result of these experiments, water-alternate-emulsion (WAE) flooding technique has been established to enhance oil recovery. The optimal ratio for the emulsion and water slug during flooding has been outlined in Chapter 4. This chapter also outlines the internal interactions involved in emulsion flooding. The results and analysis from Chapter 3 and 4 have been published as two journal papers (Baldygin et al., 2014a,b). The initial concept of the experimental system

has been presented during conference in 2012 (Baldygin et al., 2012). Results from Chapter 4 has been also published as conference paper (Baldygin et al., 2014c).

The proposed hypothesis that efficiency of emulsion flooding from well configurations was studied using 2D core holder and resulted analysis presented in Chapter 5. It includes flood front analysis, the influence of well configurations, and pressure mapping during emulsion flooding for perforation of wells oriented either vertically or horizontally. For instance, vertical injection - vertical production (VI - VP), vertical injection - horizontal production (VI - HP), horizontal injection - horizontal production (HI - HP) well configuration are tested. The 2D core holder allowed flow visualization through a single transparent side of the holder and to collect pressure data using pressure ports on another side for future pressure mapping. Using inspectional analysis, results are interpreted for reservoirs, so that industry can estimate costs and prognoses effects from emulsion flooding on oil recovery.

Chapter 6 summaries the work that has been done in current research project and also provides recommendations for future studies that can be undertaken in a field of emulsion flooding.

Additional details about the 1D and 2D core holder design procedure are given in Appendix A-1. An error analysis for equipment involved during experiments and post-experimental procedures is included in Appendix A-2. A comparison between complete and incomplete 2D core flooding experiment is shown in Appendix A-3. Source codes which were used for image analysis and pressure mapping in Chapter 5 are given in Appendix A-4 and Appendix A-5, respectively. A complete set of drawings for core holders and software flow chart with source code are presented in Appendix A-6, Appendix A-7 and Appendix A-8, respectively.

Chapter 2

Oil recovery background

The physics of fluid flow through porous media has been studied for a long period of time. Porosity, permeability, relative permeability, capillary pressure, fluid saturations and mobility ratios need to be understood, as these parameters represent relevant rock properties (Darcy, 1856; Athy, 1930; Towler, 2002). Also, it is important to know the nature of oil and water content within the reservoir. In order to be consistent in the core flooding experiments with emulsions, understanding of physical properties and the type of emulsion in use should be considered. Parameters that influence the production of emulsions with distinct functional properties and experimental procedures that effect on results are discussed below.

2.1 Porosity of Sand Pack

Porosity (ϕ) of sand pack depends on a packing density. Total bulk volume (V) of the pack is a sum of grain volume (V_s) and pore volume (V_p). Pore volume indicates the amount of void space of the rock. Porosity can then be defined as (Athy, 1930):

$$\phi = \frac{V - V_s}{V} = \frac{V_p}{V} \tag{2.1}$$

Porosity plays an important role in core flooding experiments, as OOIP occupies only the pore volume. Also, it is important to indicate effective pore volume (V_{eff}), as void space (V_v) that is connected to each other forming flow channels (Towler, 2002; Edward W. Washburn, 1921). In general for experimental work it is important to be consistent in packing of the core by using sand with the same grain density (ρ_g) and grain size distribution from experiment to experiment in order to compare results. Once it is proved that output results are valid; sand can be replaced for future studies.

2.2 Permeability of Sand Pack

In 1859, Darcy performed experiments utilizing sand packs with different orientations (Dake, 1998, p. 100),(Darcy, 1856). The pressure loss was measured across the pack with known flow rate for the given fluid. Consider a general case when sand pack can be oriented either vertical or horizontal and pressure at the inlet and outlet is measured. Taking into account that sand pack can be offset from a reference ground level the following equations for hydraulic head can be written:

$$H_{in} = z_{in} + \frac{p_{in}}{\rho g} \tag{2.2}$$

$$H_{out} = z_{out} + \frac{p_{out}}{\rho g} \tag{2.3}$$

where H_{in} and H_{out} are hydraulic head at the inlet and outlet, respectively; z_{in} and z_{out} are elevations for porous media at the inlet and outlet, respectively; p_{in} and p_{out} water pressure measured by pressure sensors at the inlet and outlet, respectively.

Taking geometry and measured flow rates, Darcys equation is defined as:

$$Q = -K\frac{(H_2 - H_1)}{\Delta l} \cdot A \tag{2.4}$$

where K is the hydraulic conductivity, which depends on fluid and porous media, Δl is the length of porous media, A is the cross section of porous media and Q is discharge from the porous media. Dividing left and right parts of the equation by cross section area, the Darcy flux (q) can be defined as:

$$q = -K\frac{dH}{dl} \tag{2.5}$$

Hydraulic conductivity can differ for different sand packs. Consider a homogeneous porous media, where the hydraulic conductivity is constant throughout the sand pack such that $K(x, y, z) = C$. Otherwise, it is required to introduce a matrix for hydraulic conductivity for three axis, as:

$$\mathbf{K} = \begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{pmatrix} \tag{2.6}$$

Consequently, for three dimensional porous media equation for Darcy flux in general form would be:

$$\bar{q} = -\begin{pmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{pmatrix} \begin{pmatrix} \frac{\delta H}{\delta x} \\ \frac{\delta H}{\delta y} \\ \frac{\delta H}{\delta z} \end{pmatrix} \tag{2.7}$$

In the current studies, two different sand packs are used. A 1D horizontal pack with the assumption that there is no effect from gravity forces and a 2D pack with

the absence of penetration in the third direction. Also, the following assumptions for Darcys law are made:

- the media is homogeneous
- the fluid is incompressible
- the flow is single phase
- the flow is happening in isotropic media

Darcys law can also be written for 1D in x-direction in the following way:

$$q = -\mathbf{K}\frac{dH}{dx} \tag{2.8}$$

or

$$q = -\frac{kA}{\mu}\frac{dp}{dx} \tag{2.9}$$

where k is absolute permeability for the porous media. For 2D, the third components for z-direction vanishes:

$$\bar{q} = -\mathbf{K}\begin{pmatrix} \frac{dH}{dx} \\ \\ \frac{dH}{dy} \end{pmatrix} \tag{2.10}$$

The equations written above are valid for single phase flow only. However, it is possible to apply Darcys law to multiphase flow. To do this it is necessary to introduce a relative (k_r), effective (k_e) and phase permeabilities (k) (absolute) (Towler, 2002; Buckley and Leverett, 1941). Absolute permeability should be measured on samples which were saturated with a single phase using Darcys law for single phase which is defind as:

$$k_a = -\frac{q\mu}{A}\frac{dx}{dp} \tag{2.11}$$

where μ is the dynamic viscosity of the fluid. For instance, if the water occupies 100% of the pore volume, the measure permeability - k_{wa} is the absolute permeability to water. Same can be undertaken for other fluids, such as gas - k_{ga}, oil - k_{oa} or emulsion - k_{ea}.

An emulsion flooding experiment begins with a water saturated pack. Consequently, absolute permeability needs to be measured for pure water flooding through a water saturated sample. Once a new fluid is introduced to the system, effective permeabilities for each phase k_{we}, k_{oe}, k_{ee} would be designated. The values are going to be less than their absolute. The ratio of the effective permeability to the absolute permeability gives relative permeability, as follows:

$$k_{or} = \frac{k_{oe}}{k_{wa}}, \;\; k_{er} = \frac{k_{ee}}{k_{wa}}, \;\; \text{and} \;\; k_{wr} = \frac{k_{we}}{k_{wa}} \tag{2.12}$$

For instance for two-phase oil/water flow the relative permeability can be written as (Buckley and Leverett, 1941):

$$k_{or} = \frac{\left[\frac{q_o\mu_o}{A}\frac{dx}{dp}\right]_{S_w}}{\left[\frac{q_o\mu_o}{A}\frac{dx}{dp}\right]_{S_w=0}} = \frac{(q_o)_{S_w}}{(q_o)_{S_w=0}} \tag{2.13}$$

The error in values of relative permeabilities depends on selected absolute permeability. When three phase (e.g. water, oil and emulsion) flooding takes place, three phase relative permeability is in use. Direct experimental investigation of three phase relative permeability properties requires measuring the fluid saturation distribution along the length of the core. Advanced technologies are required to un-

dertake this. Instead of completing three phase relative permeability experiments, two sets of two phase experiments can be carried out (Stone, 1970, 1973). Two sets refer to the use of oil-water and oil-gas systems for oil, water and gas three-phase flooding. For emulsion flooding, it is an oil-water and oil-emulsion set. Based on obtained relative permeabilities, oil to water, k_{row}, in water system and oil to emulsion, k_{roe}, in emulsion system, oil relative permeability is determined for three-phase system using rough prediction as:

$$k_{ro} = k_{row} \cdot k_{roe} \tag{2.14}$$

In the future, in case of successful development of effluent analysis equipment, the simultaneous three phase flow takes place and using the obtained numbers for saturations, ternary diagrams (Corey et al., 1956) can be plotted. Wyllies correlations (Wyllie, 1961) can be taken into account to obtain an accurate results. Wyllie proposed equations for relative permeabilities in case of wetting and non wetting phases (Wyllie, 1961). Equations for water-wet system in case of cemented sandstone, vugular rock, or oolitic limestone can be possibly used in emulsion flooding considering an emulsion as a gas phase in equations.

2.3 Saturation

Fluid saturation is a ratio of the fluid volume that occupies the pore volume by a particular phase (water, oil or emulsion). Water saturation, S_w, is defined as:

$$S_w = \frac{V_w}{V_p} \tag{2.15}$$

where V_w is the pore volume occupied with water. This is also an important parameter that effects EOR performance. The number of saturations parameters present in the systems depends on the number of fluids present in the porous media. Thus, saturation of water, oil and emulsions take place in emulsion flooding named as S_w , S_o, S_e, respectively for water, oil and emulsion. The sum of the three equals to one such that:

$$S_w + S_o + S_e = \frac{V_w}{V_p} + \frac{V_o}{V_p} + \frac{V_e}{V_p} = 1 \tag{2.16}$$

During core flooding experiments, there are also several other important parameters, such as irreducible water saturation and saturation for immobile oil, to be determined as: S_{wi}, S_{or}. Irreducible water saturation is obtained during initial oil saturation process. It shows the amount of water that cannot be reduced further by displacing water with oil, displacing process. While, immobile oil shows the amount of oil that cannot be recovered from porous media at the end of oil recovery stages.

2.4 The physical Process of Oil Recovery

In order to understand the recovery process and why applied pressure is needed, consider single drop behavior in a pore as shown in Figure 2.1. The capillary force keeps oil inside the pores during secondary recovery. During secondary and tertiary recovery injected fluids are intended to overcome capillary pressure which is defined as using Young-Laplace equation:

$$P_c = \frac{2\gamma \cos\Theta}{r_{pore}} \tag{2.17}$$

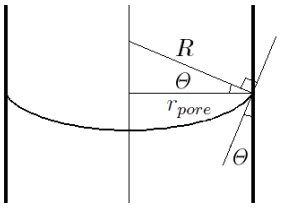


Figure 2.1: A schematic of the concept of a single pore with trapped oil (contact angle is less than 90°).

where γ - interfacial tension; Θ - contact angle; r_{pore} - is the pore radius, as defined in Figure 2.1

In the case of two pores located above the same free water level, capillary pressure, contact angle and interfacial tension will be the same. This means that for water-wet pores the amount of oil in percentage to pore size relation will be greater for large pores than for small pores (Schramm, 2006). Consequently, oil will tend to recover from small pores first for water-wet (Schramm, 2006, p. 269). The opposite situation is for oil-wet pores. Oil will tend to recover from large pores and the amount of oil is greater in small pores. A schematic of a single oil drop which is sitting on a pore is shown in Figure 2.2. In order to squeeze and move the drop through the pore (in other words recover the drop), pressure difference across the drop should be applied. (Nazzal and Wiesner, 1996) termed it as the critical pressure (P_{crit}) at which drop starts moving. From the Young-Laplace equation (capillary force) (Green and Willhite, 1998, p. 18), (Nazzal and Wiesner, 1996; Leverett, 1940), they derived:

$$P_{crit} = \left(\frac{2\gamma_{o/w}}{r^*}\right) - \left(\frac{2\gamma_{o/w}}{R^*}\right) \tag{2.18}$$

where r^* and R^* are radii shown in Figure 2.2 and $\gamma_{o/w}$ is the interfacial tension between two fluids, oil and water. Radius, r^* can be found from:

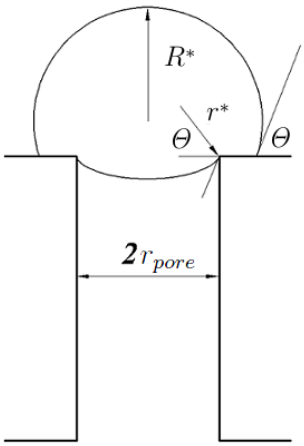


Figure 2.2: Oil drop is sitting on the throat of pore (following Nazzal et al. (Nazzal and Wiesner, 1996)).

$$r^* = r_{pore}/\cos\Theta \tag{2.19}$$

An advantage of using emulsion as an extrusive liquid is that it contains surfactants which reduce the interfacial tension between the two fluids. Thus, the critical pressure required for oil recovery is reduced and the amount of recovered oil increases.

Another important parameter for oil recovery is the capillary number (Washburn, 1921) which describes the ratio of the viscous and interfacial forces in the form:

$$N_c = \frac{\eta v}{\gamma \phi} \tag{2.20}$$

where η is viscosity of displacing fluid; v is the velocity of the displacing fluid; ϕ is the porosity; and γ is the interfacial tension. At the beginning of a water flooding process, the capillary number can be around 10^{-6} (Schramm, 2006; Moore and Slobod, 2013). When residual oil mobilization begins the capillary number

would be around 10^{-5}. Typical chemical flooding, *e.g.* presence of surfactant in flood front, regime would lie into the range from 10^{-5} to 10^{-3}. The end of oil recovery brings the capillary number to range from 10^{-3} to 10^{-2}. Consequently the capillary number changes as the remaining percentage of oil inside reservoir is lowered (Schramm, 2006; Foulser et al., 1991). Romero et al. (2011) developed a network model of flow of oil-water emulsion through porous media. It was shown that during the process of emulsion flooding capillary number ranges from 10^{-6} to 10^{-2}.

The process of emulsion flow through sand is similar to the process of oil recovery. Emulsion is a mixture of droplets of one fluid in another. Similar physics can be applied for an emulsion with assumption that emulsion is a single phase fluid.

2.5 Mobility

Mobility can be derived as a ratio of relative permeability of a fluid to viscosity of the same fluid. Using the term mobility, defined as (Homsy, 1987):

$$M_{ro} \text{ for oil phase} = \frac{k_{ro}}{\mu_o} \tag{2.21}$$

$$M_{rw} \text{ for water phase} = \frac{k_{rw}}{\mu_w} \tag{2.22}$$

$$M_{re} \text{ for emulsion phase} = \frac{k_{re}}{\mu_e} \tag{2.23}$$

The mobility ratio is another important parameter defined as a ratio of oil mobility to water mobility for water-oil system:

$$\lambda = \frac{M_{ro}}{M_{rw}} \tag{2.24}$$

or as a ratio of oil mobility to emulsion mobility for emulsion-oil system:

$$\lambda = \frac{M_{ro}}{M_{re}} \tag{2.25}$$

If it is less than or equal to one, the process of oil recovery is characterized as "stable displacement" (Homsy, 1987; Schowalter, 1965). For mobility ratio greater than one, the process of oil recovery suffers from unstable displacement and other processes such as fingering can occur (Homsy, 1987; Schowalter, 1965). Mobility control is a key component for effectiveness of oil recovery. French et al. (1986) demonstrated that temperature and water dilution resistant emulsion with appropriate size can be formed and used for blocking mechanism. It allows to modify permeability of the porous media and improved mobility control. Hence, the mobility of emulsion can have a wide range and be matched for a given sand pack (Speight, 2006).

2.6 The Composition of the Fluids

The ultimate goal for core flooding experiments is scaling to the real reservoir. Results obtained in experiments should correlate to oil recovery in a field. Thus, it is important to take into account for the composition of oil and water and other important species content.

Fluid composition also affects oil recovery and emulsion stability. The water present in sandstone reservoirs has major cations and anions, such as Na+ and Cl- (Schmidt, 1973). Their content can vary with a depth with a rate of 30,000 mg/l Na+ per 1,000 ft and 45,000 mg/l Cl- per 1,000 ft down to 3,000 ft (Schmidt, 1973). Bagci et al. (2001a; 2001b) reported the results of brine flooding tests using a core flooding system and showed that brine composition affects oil recovery. The

increased amount of salt reduces the amount of recovered oil from reservoir oil. Higher pH can be achieved with lower salinity for brine composition. The pH level was stabilized at 7.2, 7.4 and 6.8 for 2 wt %, 5 wt % and 10 wt % $CaCl_2$, respectively after 1.0 pore volume of injection (Bagci et al., 2001a). Yildiz et al. (2013) identified that 16 % difference in an oil production can be as a result of using different brine compositions, e.g. Brine 1 (4 % NaCl + 0.5 % $CaCl_2$) is more efficient than Brine 2 (2 % $CaCl_2$). Also, in consolidated/unconsolidated limestone samples there may be a reduction in permeability when the critical salt concentration in brine is achieved (Bagci et al., 2001a).

The stability of emulsion also depends on fluid composition. Moradi et al. (2011) studied the stability of water-in-oil emulsion prepared with a crude oil and a synthetic reservoir water. It was shown that emulsion becomes unstable at higher salinity (1, 10 and 50 % water salinity values were examined). The base 100 % solution was a composition of NaCl, $CaCl_2$, Na_2SO_4, $MgSO_4$ and TDS.

2.7 Production of the Emulsion

The process of emulsion production is termed emulsification and contains different physical processes. Walstra (1993) divided the process into three main categories: deformation coupled with possible break up; adsorption and transportation of surfactant to newly developed droplets; and bouncing and merging of drops. For the petroleum industry, there are two options to create emulsions *in-situ*, inside the reservoir, or *ex-situ*, outside in a process facility. Direct injection of alkali/surfactant (AS) solution after water flooding into porous media can result in emulsion formation (Kumar et al., 2012). Salt concentration and alkali concentration affect emul-

sion formation (Kumar et al., 2012; Perles et al., 2012). Another method is to use higher velocities and more efficient emulsifiers to increase shear stresses *in-situ* and form emulsion (Sarma et al., 1998).

Outside process facilities can also be used to prepare emulsion using oil, water and surfactants. Warmed heavy oil can be added to hot water premixed with surfactant and mixed using an industrial mixer. Another way would be circulating the fluid through a jet at a set volume rate with a centrifugal pump outside of reservoir (McAuliffe, 1973). Another procedure, the aqueous-phase solution and oil-phase solution can be prepared separately and stirred using a homogenizer (Son et al., 2014).

During production, the emulsion can be specified to be stable or unstable. Stable emulsions have a constant drop size distribution, while unstable emulsion forms larger droplets. In the critical case, the two liquids can segregate and form creaming and generate flocculation (Schramm, 2006). In the case of drop size around 10-100 nm, drops are stable (Windhab et al., 2005; Binks, 1998); however for larger sizes coalescence can occur. Surfactants can be added into the mixture to reduce surface tension between oil and water to avoid droplet break-up for sizes in the range of 0.5 to 100 micrometers (Binks, 1998). In addition, McAuliffe (1973) pointed out that mineral surfaces, reservoir water salinities, and water hardness can break droplets. Mandal et al. (2010a) observed the influence of stirring speed of a three-blade propeller and time of mixing on a stability of emulsion. In the present studies, the emulsion is prepared prior to experiment with known drop size distribution viscosity, stability and water/oil ratio (WOR).

It is important to know and control the emulsion parameters. During emulsion

production, the energy input can indicate different emulsion properties. In order to connect the energy input to properties of the liquids, such as viscosity of the continuous phase and interfacial tensions, the Weber (*We*) number can be used and is defined as (Schramm, 2006, p. 59):

$$We = \frac{\eta_1 \dot{\gamma} R}{\gamma_{12}} \tag{2.26}$$

where η_1 the viscosity of the continuous phase; $\dot{\gamma}$ shear rate; R droplet radius and γ_{12} the interfacial tension. Values of *We* in laminar and turbulent flows are discussed in detail in (Walstra, 1993) and can be used during emulsion production. Dependence of *We* versus ratio of dispersed phase velocity and continuous phase velocity is shown in (Schramm, 2006).

The production an emulsion is not part of this project. However, understanding of this process is necessary for future reference during experiments. Quadrise Canada Corporation, a collaborator in the project supplied an emulsion for this project. Their emulsions manufactured under brand name E^2EOR are stable and flexible in parameters. Their technology allows to tailor parameters for any particular application, such as viscosity, hydrocarbon content, the drop size distribution, chemistry and stability. The drop size can be controlled within a few hundred nanometers to less than 10 μm. The amount of available emulsion was limited due to the closing down of the company in earlier 2012 resulted in a reduced number of experiments being carried out to test the hypothesis.

2.8 Scaling Analysis

Carrying out the experiments in the field can be expensive and sometimes is even impossible due to lack of the information about reservoir conditions. Core flooding experiments simulate reservoirs porous media with assumptions stated by researchers (similar to Darcys law assumptions). Based on the obtained results, the number of assumptions can be reduced to make the porous media closer to the reservoir conditions. During core flooding experiments different core holders are in use. Core holders can be 1D, 2D or even 3D. The experiments are performed using pre-set physical parameters. The ones obtained from experimental results should be presented in a way that makes them applicable to the actual field reservoir. Thus, these parameters and dimensions of a core holder have to be scaled from typical reservoir where the method being developed will be used. The dynamic and physical characteristics of the experiment and reservoir are required to be scaled up.

For instance, pressure and temperature are the same in both cases. However, the dimensions of the cells and flow rates of injected liquids are parameters for scaling procedure. In order to set the flow rate for the 1D core holder, the Willhite's scaling coefficient is used (Willhite, 1986). Empirical correlation for water flooding to estimate the stabilizing criteria has been developed using the dimensional analysis by Rapoport and Leas (1953). According to Bagci et al. (2001b, p. 90), "displacement data for linear floods were independent of length, rate of injection, and viscosity of the injected water when the numerical value of the scaling coefficient, $Lu_t\mu_w$, exceeded a critical value", which is given as

$$Lu_t\mu_w = 5.85 \times 10^{-9}N \tag{2.27}$$

where u_t - total Darcy velocity in linear displacement, m/D (ft/D); μ_w - water-phase viscosity, Pa·s; L - length of cell, m. In reservoir engineering, it is a common practice to use ft/D as a unit for the velocity of flood front. After determining all the variables with fixable units, the following equation was obtained:

$$q = \frac{0.00351 \cdot A}{L \cdot \mu_w}, \text{cm}^3/\text{min} \tag{2.28}$$

where d - diameter in 1D cell, cm; A - cross sectional area for 1D cell, cm^2. It was found that applicable flow rates for 1D core holder are within a range of 0.1 to 1 cm^3/min. For the 2D core holder it is within a range of 1 to 1.75 cm^3/min.

Chapter 3

A new laboratory core flooding experimental system [1]

3.1 Introduction

The oil and gas industry heavily relied on laboratory core flooding experiments, where a core pack, often unconsolidated sand particles (Santosh et al., 2007; Hadia et al., 2007b; Mandal et al., 2010b; Hadia et al., 2008a) or in some cases consolidated cores (sandstone and carbonate outcrops) (Hadia et al., 2007a; D'Elia-S and Ferrer-G, 1973; Hadia et al., 2012, 2008b), mimicking the porous natural reservoir, is traditionally flooded with an injection fluid and resultant output in terms of oil recovery is measured. Existing core flooding systems use this concept, which was introduced for the first time in early 1930's (Wyckoff and Botset, 1936). When the industry is exploring new options for flooding the existing oil fields abandoned after the primary recovery process (Tzimas et al., 2005), there is a need to produce more robust, automated system to reduce time associated with experimental and post-experimental analysis.

A critical solution is provided to this pressing need for the oil and gas indus-

[1] A version of this chapter has been published. A. Baldygin, D. S. Nobes, and S. K. Mitra, "A new laboratory core flooding experimental system", *Industrial & Engineering Chemistry Research*, DOI: 10.1021/ie501866e, 2014

try by providing a robust, versatile, precise and automated core flooding platform which can be tested for various types of reservoir conditions using some of the advancements in fluid controls and data processing and automation. For validation purposes for the developed system, initially experiments are conducted with water and paraffin oil only, water flooding. To anticipate the future performance of the core flooding system for potential enhanced oil recovery (EOR) studies, applicability of the system for flooding complex fluids, such as an emulsion, is also demonstrated here.

3.2 Traditional Core Flooding Systems

The traditional core flooding system, as described in the existing literature (Hadia et al., 2007a; Mandal et al., 2010b), has three main sections: the upstream, the core block, and the downstream. The upstream supplies relevant fluids (saturation and flooding agents) to the core block typically using syringe pumps and piston accumulators (PA) to provide a controlled flow rate. The core block contains the porous media and often simulates the reservoir conditions which could include overburden pressure and a thermal jacket to maintain reservoir temperature. The downstream collects the effluent from the core block by using fraction collectors (Nasralla et al., 2011) or two-/three-phase separators (Boye et al., 2008; Legowo and Pratomo, 1999), the analysis of such data often leads to the estimation of the efficiency of tested oil recovery method.

Components of the system are connected to a data acquisition module through which the user can communicate, which is referred here as a control module. Such experimental techniques work well for single phase fluids (such as water or oil)

and certain restricted two-phase systems (oil and water mixture as effluent from the production wells).

Table. 3.1 provides a summary of existing core flooding systems, used for academic research and industrial applications for research in oil and gas industry. It can be noted that most of the systems use syringe pumps coupled with piston accumulators (Samanta et al., 2012; Hadia et al., 2007b; Legowo and Pratomo, 1999; Nobakht et al., 2007) which limits the type of liquids that can possibly be injected into the core as the inner surface of the piston-cylinder assembly of the syringe pump is susceptible to corrosion. The piston accumulator size limits the volume of fluid that can be continuously injected during flooding experiments and *a priori* calculations need to be performed to determine the size and specification of the piston accumulator to undertake a given core flooding study.

Table. 3.1 also suggests that a fraction collector was used for the downstream section in significant amount of the core flooding systems, which resulted in discrete estimation of oil and water effluents for the recovery curves (Samanta et al., 2012; Hadia et al., 2007b; Mandal et al., 2010b). Also, post experimental analysis is required to analyze the fraction of each effluent phase collected in each tube (Mandal et al., 2010a; Hadia et al., 2007b) of the fraction collector. In some cases, an ultrasound separator has been used for the effluent collection and analysis (Legowo and Pratomo, 1999). It is an effective method for experiments with three-phase effluent fractions, where phases are distinctly different from each other. As an example, experiments which involve water/oil/gas flooding has three components that can be separated in effectively inside the column of the three-phase separator using ultrasound and later can be quantified by volume.

Table 3.1: Summary of existing core flooding systems used globally.

Author	Year	Upstream	Downstream	Used fluids
Hornof and Morrow	1987	Metering pump	n/a	Isooctane, brine, water and oil
Chang and Grigg	1994	Syringe pumps with PA	Multiport rotary valve with vials	CO_2, brine, oil and surfactant solution
Legowo and Pratomo	1999	Displacement pump ”Quizix” with PA	Fraction collector and separator	Water, oil and different microbial cultures
Bagci et al.	2001	Syringe pump	Fraction collector	Water, oil and different microbial cultures
Sedaee and Fariborz	2004	Displacement pump with PA	Effluent condenser with collector	Steam, CH_4 and heavy oil
Nobakht et al.	2007	Syringe pump with PA	Oil sample collector with gas flow meter	Crude oil and CO_2
Hadia et al.	2007	Dual piston syringe pump with PA	Fraction collector	Oil and water
Mandal et al.	2010	Syringe pump with PA	Fraction collector	Oil-in-water emulsion, oil and water
Samanta et al.	2012	Syringe pump with PA	Fraction collector	Oil, brine and chemical slug
Veerabhadrappa et al.	2013	Syringe pump with PA	Effluent sample collector	Oil, water and polymer
Ko et al.	2014	Syringe pump with PA	Fraction collector with centrifuge tubes	Brine, oil and surfactant

These traditional core flooding systems need to be modified in order to make it more versatile in terms of using a wide range of flooding fluids such as surfactants, polymers and emulsions. Often in such scenarios, to calculate the various fluid volume fractions in real-time is needed which requires the inclusion of modern fluid handling systems to increase the functionalities of traditional core flooding system.

3.3 Developed Core Flooding System

A core flooding apparatus has been designed and constructed by taking into account the potential challenges related to a traditional core flooding system and the availability of new technology. This is a further modification to the initial conceptual design reported earlier (Baldygin et al., 2012). A schematic of the system is shown in Figure 3.1. This schematic shows the layout for the core flooding system, which is divided into three sections: the upstream, the core block and the downstream. This schematic was developed using diagramming software (Microsoft Visio, Microsoft, Co.) with a set of standard diagram elements. The fluid lines, control and data transfer lines are shown to guide the reader through the fluid and data flow.

The upstream side of the system provides continuous injection of three different types of liquids. It consists of fluid holding containers (carboys), pumps, and mass flow meters. Three carboys (E-1,...,E-3) are directly connected to three dual piston pumps (Smartline 100, Wissenschaftliche Gerätebau Dr. Ing. Herbert KNAUER GmbH) through 6.35 mm (1/4 in.) stainless steel tubes. The precision and accuracy of the upstream pumps at 12 MPa and for flow rate of 1 ml/min are equal to $<$ 0.5 % and $<$ 1.0 %, respectively. These numbers are close to the flow accuracy of

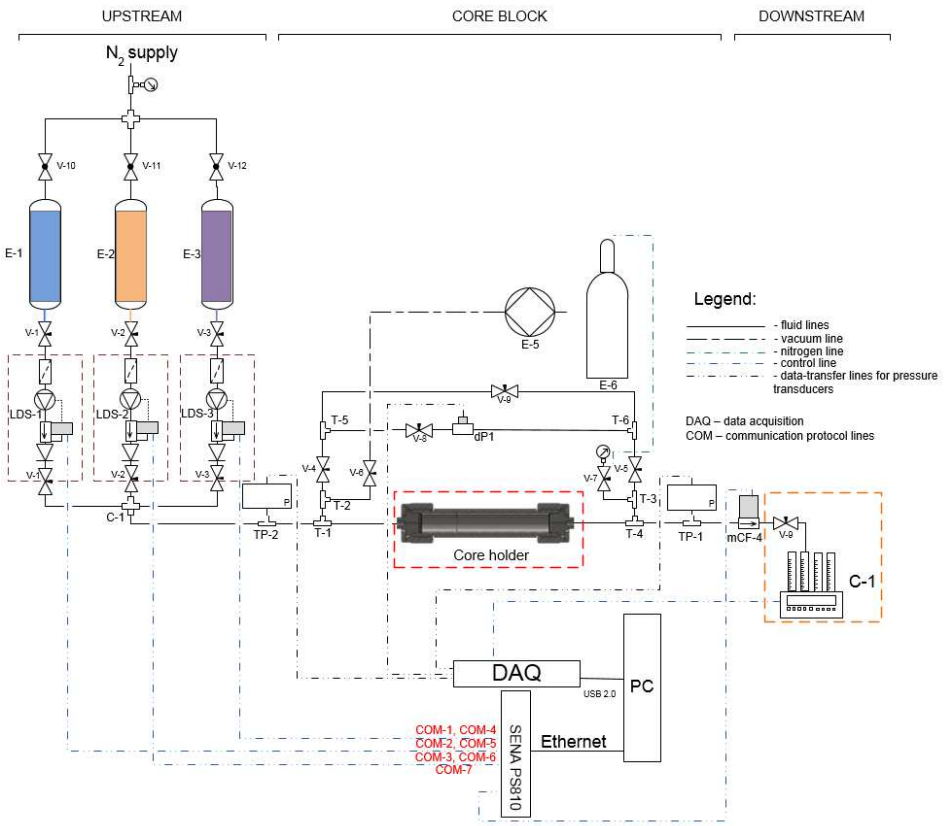


Figure 3.1: Schematic of the core flooding experimental apparatus.

a syringe pump. For example, 500D series pump (500D Teledyne Isco) has flow accuracy equal to 0.5 % of set point. In order to avoid cavitation at the inlet side of the pumps, the carboys are pressurized with nitrogen at 0.138 bar(g) (2 psig). Mass flow meters (M13, Bronkhorst Cori-Tech BV) are installed downstream of the pumps. Two of the mass flow meters are built of stainless steel, while one is made of Hastelloy to allow the use of salt (brine) water. According to the calibration certificates of the mass flow meters, the measurement error is $\pm$ 0.2% of the full-scale reading + 0.5 g/h. During the factory calibration procedure, density deviation was found to be 0.11 kg/m^3, -0.16 kg/m^3, -0.27 kg/m^3 and 0.01 kg/m^3 for water, oil, emulsion and downstream mass flow meter, respectively. The density measured using the three mass flow meters was compared with the one calculated using software (Bronkhorst High-Tech FLUIDAT©, Bronkhorst Cori-Tech BV) which defines physical properties of a fluid based on specific fluid temperature and pressure conditions with theoretical calculation methods. For example, with the water flow rate of 0.5 cm^3/min for half hour (approximate time to fill a single test tube at the fraction collector) the error is $\pm$ 0.28 ml considering density of water at experimental conditions. Together with the pumps are coupled to check valves, they form the liquid dosage systems (LDS-1,...,LDS-3), which can keep constant mass/volume flow rate at a pre-set pressure.

There are three LDS in the current configuration to deliver water, oil and emulsion into the core block as needed. The LDS system can work simultaneously if it is needed for certain experimental conditions. In addition, the upstream has open access to refill the carboys to allow continuous injection of liquids for lengthy experimental runs. There is no need to stop an ongoing experimental run for refilling

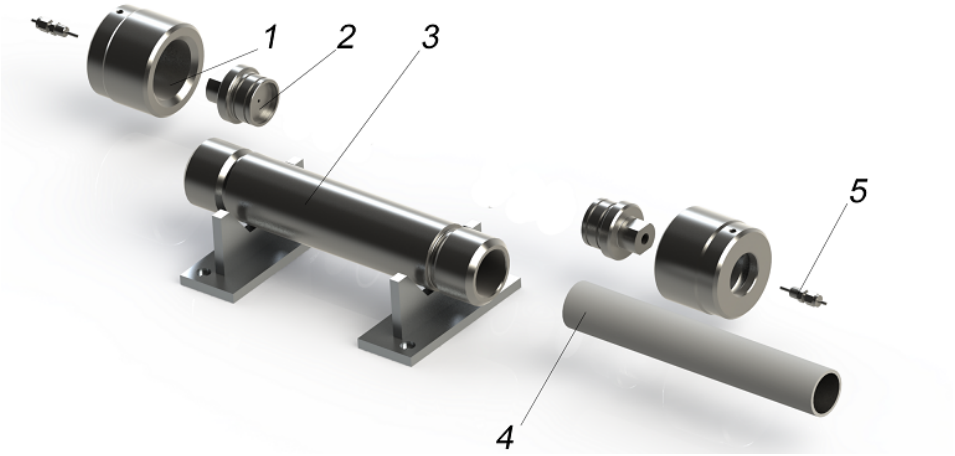


Figure 3.2: Exploded view of the core holder (1 - cap, 2 - end plug's body, 3 - body, 4 - internal surface, 5 - fitting 1/8 NPT to 3.175 mm (1/8 in.) tube).

purpose, which is a shortcoming in case of a traditional system with piston accumulators at the upstream side.

The core block consists of the core holder, in which the porous medium is usually packed which can be sand pack or samples, inline pressure sensors and differential pressure sensors. A biaxial horizontal core holder, shown in Figure 3.2, is used for containing and packing the medium. The core holder, shown here, is designed to perform one-dimensional core flooding experiments. However, any other two-dimensional (Santosh et al., 2007) or three-dimensional (Hadia et al., 2007b, 2008a) core holders, as and when available, can be plugged into the system through appropriate connections to 3.175 mm (1/8 in) stainless steel tube. The dimensions of the core holder used here as an example are: length: 284 mm (11.18 in) and diameter : 38.1 mm (1.5 in). This design incorporates minimal fluid leakage between the inner surface of the coreholder and the sand pack. At the same time, it also reduces the dead volumes at the inlet and outlet sides of the sand pack due to the modified entrance region, shown in Figure 3.3. The entrance region incorporates

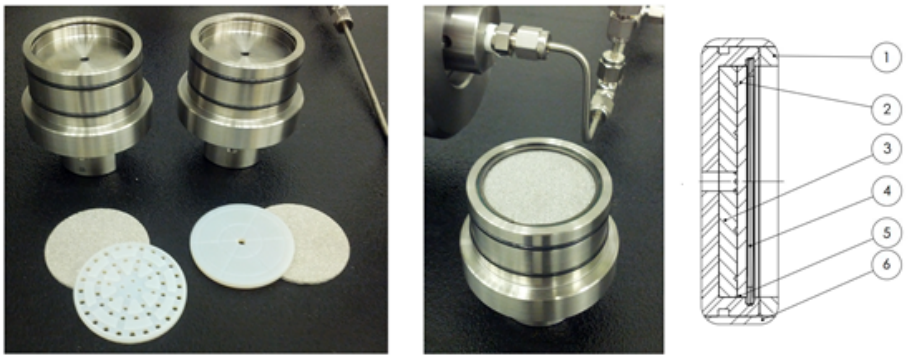
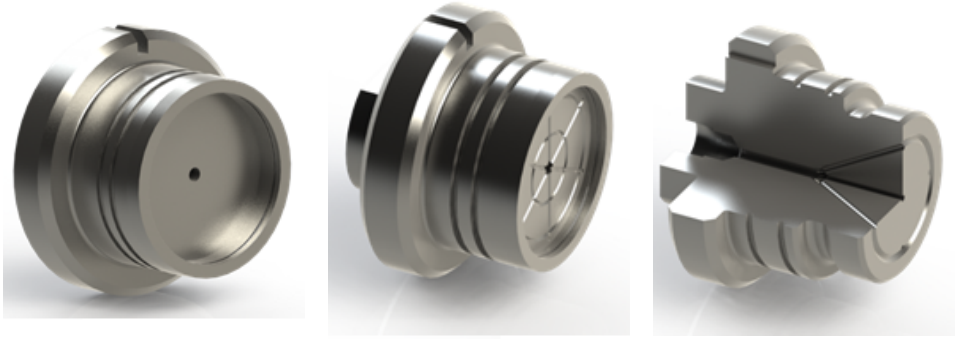


Figure 3.3: End plugs of the biaxial core holder. (a) - isometric view of end plugs; (b) - end plug assembled; (c) - cross section view of single end plug (1 - sandblasted internal surface, 2 - stainless steel filter, 3 - 3D-printed plate, 4 - strain ring, 5 - end plug's body, 6 - core holder's body

modifications to the patents of Stanley (1984; 1985). It combines the geometrical configurations of both patents and thereby replaces/modifies the distribution plates as per current needs. A sand-blasted plastic tube is installed inside the core holder to prevent such fluid leakage. The distribution plugs, located at the two ends of the core holder, have built-in stainless steel media with mesh size of 40 μm and 1.59 mm (1/16 in.) thickness and distribution plate printed using a 3D printing. This combination redistributes the liquid across the cross-section and reduces dead volumes.

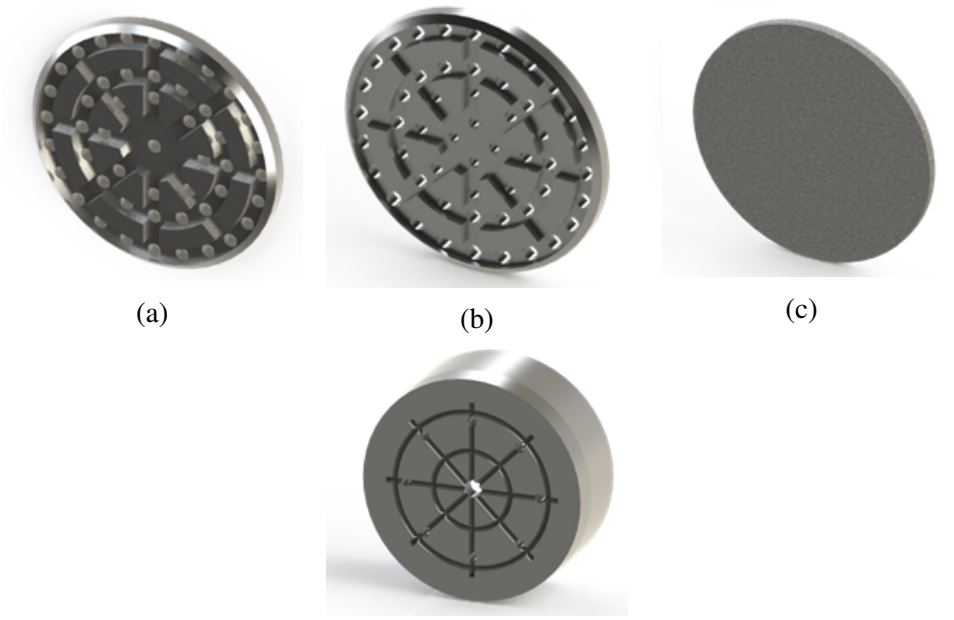
The design for the distribution plates was optimized by performing separate simulation using commercial software (SolidWorks Simulations, Dassault Systemes SolidWorks, Corp.). In total, nine different configurations have been analyzed as a combination of three configurations for the end plug, which are shown in Figure 3.4, and four configurations for the distribution system, which are shown in Figure 3.5. Design of the end plugs shown in Figure 3.4 (b) and (c) bases on industry standard available in a core holders from two main manufactures (Core Laborato-

ries (www.corelab.com, 2014a,b) and CORETEST Systems Inc. (www.coretest.com, 2014)). Stainless steel filter, shown in Figure 3.5 (c), forms artificial porous media after perforated plate, shown in Figure 3.5, and helps additionally redistribute liquid before it goes into a sand pack. In addition, it provides sand control in a core holder. The cap with 9 holes, shown in Figure 3.5 (d) has internal thread and can be installed only on end plug with channels for flow redistribution. Thread allows to change orientation for holes with respect to the open channels and twist flow internally.



(a) (b) (c)

Figure 3.4: Variation of geometric configuration for end plug (a) single entrance, flat surface; (b) single entrance, groves for flow distribution; (c) single entrance, channels for redistribution.



(a) (b) (c)

(d)

Figure 3.5: Variation of sub-components for end plug (a) perforated plate, single hole; (b) perforated plate, four entrance holes; (c) 40 micron stainless steel filter made of 316L SS; (d) cap with 9 holes and channels.

A comparison of the flow distribution between traditionally used and improved configuration is shown in Figure 3.6. The color map represents internal velocity through the cross section of the sand pack. Once the configuration was finalized particular attention was given to the size of the holes in the radial direction (Stanley, 1984). It was optimized to guarantee uniform flow distribution through the porous medium with known range of permeability and porosity.

For pressure monitoring, there are three pressure sensors (FP2000 series, Honeywell International Inc.), TP-2, TP-1 and dP-1, to measure the inlet pressure at the upstream, the outlet pressure at the downstream, and differential pressure across the porous media.

The downstream section has an effluent management system which allows it to quantify different fractions of effluents removed from the core block. The mass

(a) Traditional configuration for the end plug

(b) Improved configuration for the end plug

Figure 3.6: The horizontal velocity component distribution in the axial cross-section of the core holder obtained with different configurations of end plugs.

fraction of each component needed to be calculated to estimate the amount of oil recovered, which is often the ultimate goal for such core flooding system. The effluent management system has a mass flow meter and a fraction collector (CF-2, Spectrum Laboratories, Inc.). The fraction collector uses graded test tubes with 15 ml volume each. The error in a measuring volume of liquid collected in a test tube (430055, Cole-Parmer Canada Inc.), used for checking the effluent fraction data, is $\pm$ 0.25 ml. Since this error would be accumulated from sample to sample, the cumulative error in the oil production is around 6.5 ml (estimated based on expected number of test tubes in 1D experiments). The error bars incorporated for each data point in the recovery curves represent the error in measurements. Complete error analysis can be found in Appendix A-2. The mass flow meter (M13, Bronkhorst Cori-Tech BV) has the same specifications as that used for water flow, installed in LDS-1. The mass flow meter provides the flexibility to deal with effluents having different densities and it allows automatic evaluation of volumes of different effluent phases, the details of which are described later.

The experimental setup can be pressurized with nitrogen, E-6, and tested for leakage using soap-water, usually undertaken without the porous sand pack in place.

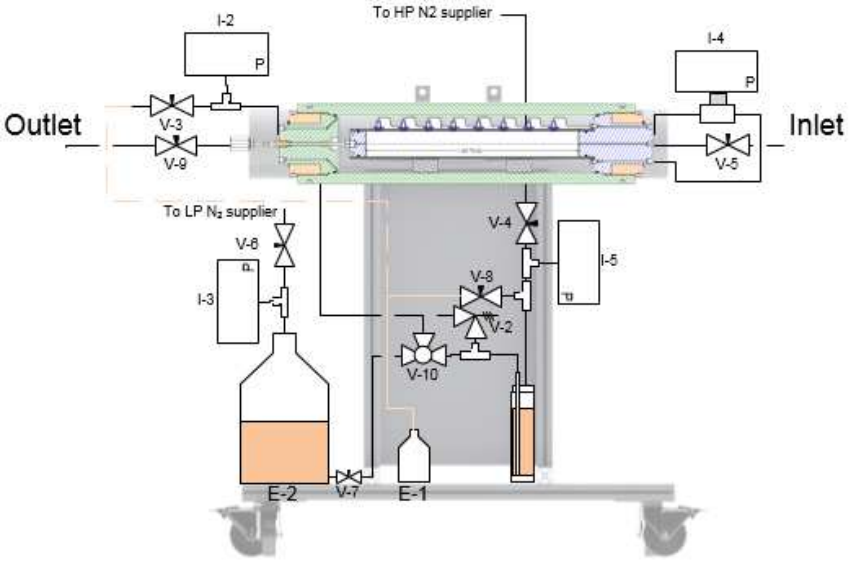
3.3.1 Core Flooding System Flexibility

The advantage of the developed system is that it is designed as a ”plug-and-play” modular system. There is a possibility for further modification of the core block, which is only limited by the specification of the upstream pumps. The region highlighted with red dashed square in Figure 3.1 can be replaced with the sub-system, described in Figure 3.7(a), quite readily, or with a 2D core holder, used for experi-

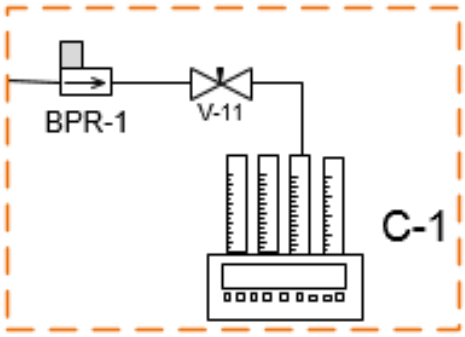
ments in Chapter 5.

Example of the subsystem, proposed in Figure 3.7(a), has a Hassler core holder (www.corelab.com, 2014b) which allows the core to be maintained with a confining pressure up to 10,000 psi, to use actual cores previously extracted from reservoirs (outcrops) and to measure pressure drop across the core using eight tap-points. The confining pressure in the present configuration can be maintained using a high pres-sure N_2 line. This further can be replaced with either high pressure hand pump or a syringe pump. The temperature inside the core (or along its outer surface) can be measured using any of the eight tap-points of core's sleeve via thermocouples installed through the ports located on the end plug with appropriate pressure sealed feedthroughs (Series WF, Spectite Inc.). In order to maintain the temperature of the core block similar to reservoir conditions, the core holder can be placed inside an environmental chamber or an ultra-thin heat sheets can be mounted at the outer sur-face of the core holder covered with fiberglass pipe insulation. The chamber allows to maintain temperature up to 200 °C, whereas the heat sheets allow to maintain temperature up to 100 °C. The temperature can be controlled via programmable temperature PID controller with feedback from one or more thermocouples.

Similarly, the downstream section can be also modified to satisfy the requirements for the inline pressure. For example, a digital back pressure regulator (EL-PRESS P-712CV, Bronkhorst Cori-Tech BV) can be installed between downstream mass flow meter and fraction collector, as shown in Figure 3.7(b). It would allow to set the back pressure up to 103.4 bar(g) (1500 psig). These modifications to the core block and downstream section of the core flooding developed allow end-users to obtain the required parameters to mimic desired reservoir conditions.



(a) Core block



(b) Downstream section

Figure 3.7: Schematics of the further updates to the developed system.

3.3.2 Core Flooding System Limitations

One of the limitations of the current system is that the produced water cannot be separated from the emulsion and re-injected back to the core block as a flooding fluid for further oil recovery. In order to utilize such produced water back into the flooding system, it would be required to install additional components at the downstream to separate emulsion, paraffin oil and water. Such separation of oil/water emulsion can be done using traditional membranes (Chen et al., 2009) or more recently through the use of microfluidics devices (Fidalgo et al., 2008).

3.3.3 Remote Control

In order to minimize human intervention and have robust control on the experiment, software with a graphical user interface (GUI) was created for the entire system using a commercial environment (LabWindows CVI, National Instruments Inc.). This software allows user to control and track different components during the experiment, save data and analyze various experimental conditions. A commercial data acquisition (DAQ) system (cDAQ-9172, National Instruments Corp.) with onboard control modules for pressure sensors and fraction collector is used for data acquisition. In addition, the mass flow meters and pumps are connected to a universal eight port device server (PS 810, Sena Technologies, Inc.) which transfers and receives signals from personal computer (PC) through an ethernet network.

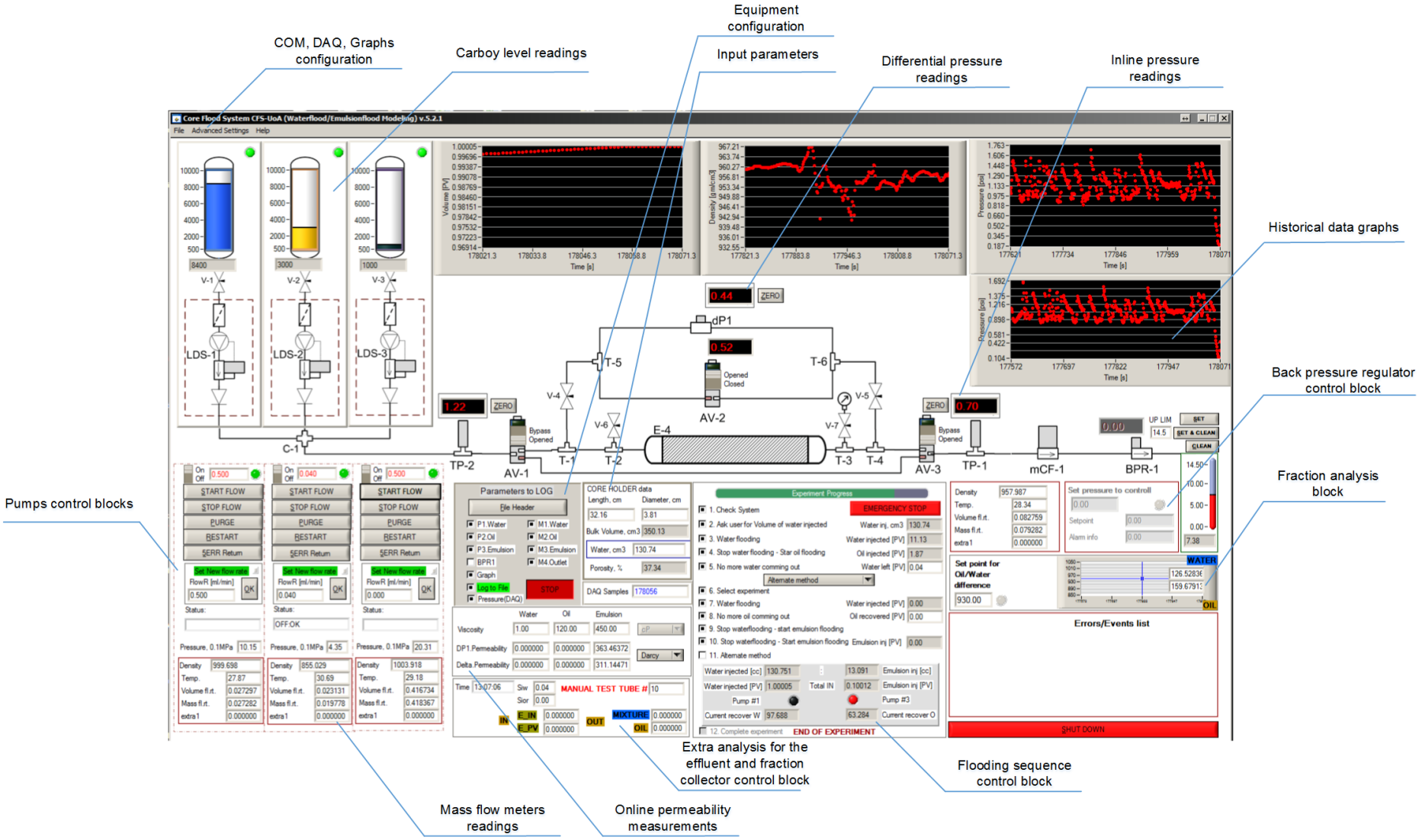


Figure 3.8: A graphical user interface to perform automation for the core flooding system.

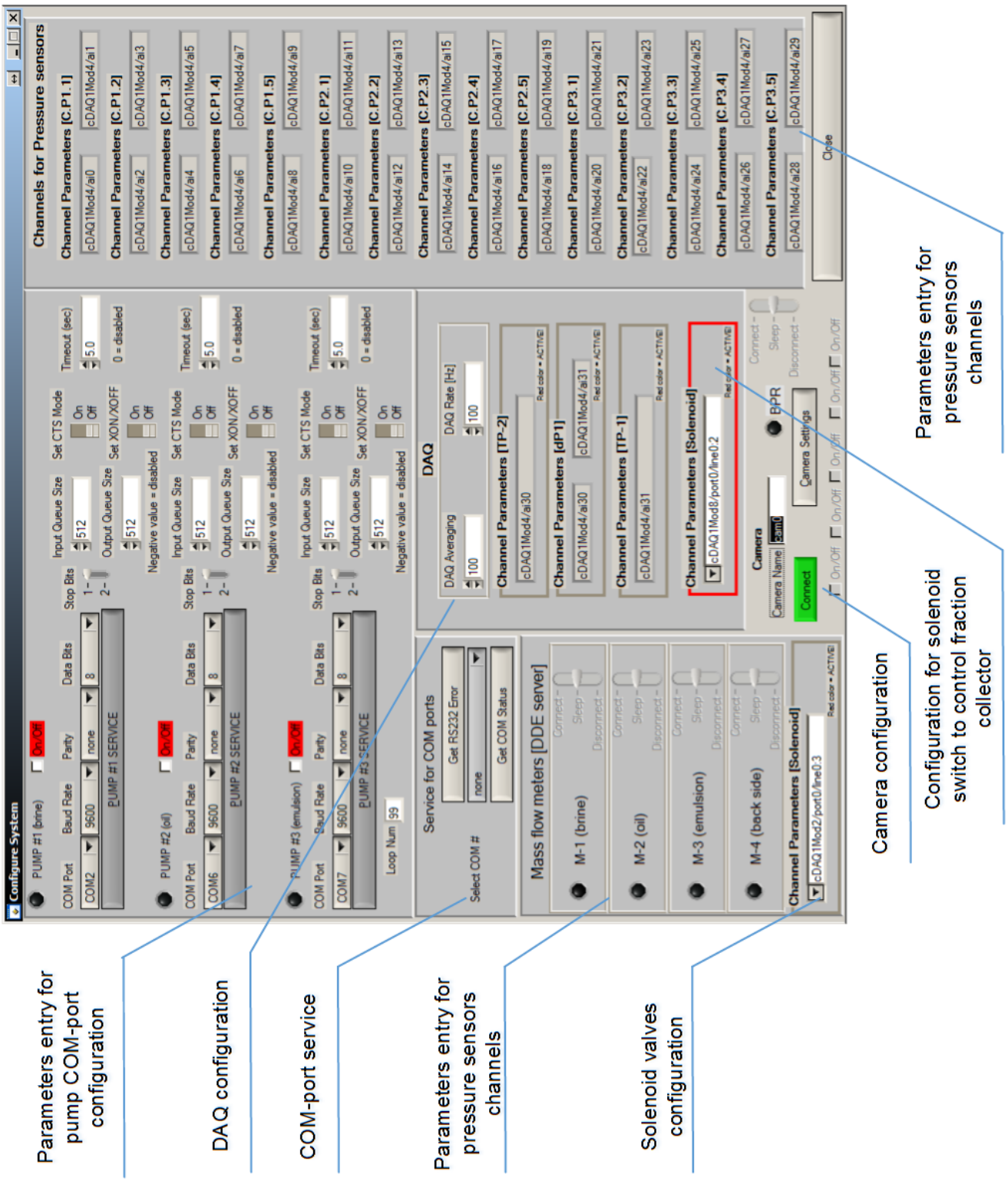


Figure 3.9: A graphical user interface to perform configuration for the core flooding system.

The developed GUI interface is shown in Figure 3.8. Similar to the experimental setup, it can be divided into three sub-sections: the upstream, the core block and the downstream. There are components for controlling different processes of experiment, load/save configuration parameters, saving data, monitor effluent properties, pressure and other parameters involved in each run. The developed software allows to monitor system in real time. There is a possibility to run experiments in "auto" operating mode using the developed logical structure. Figure 3.9 shows the GUI which was developed to configure individual components of the system.

3.4 Materials and Specifications

Using the developed core flooding system, both traditional core flooding using water as a flooding agent (secondary recovery process) and emulsion flooding using *ex-situ* produced emulsion (enhanced oil recovery) were performed. In this section, the details of different materials used during this experimental validation process for the core flooding system along with standard data extraction in the form of porosity and permeability of the core pack used for each flooding experiment are provided.

During 1D core flooding experiments, the injection rate was set at 0.5 cm^3/min. The core flooding temperature was at the room temperature of 22°C for all experiments. Distilled water (DI) produced from a water purification system (PURELAB Ultra, Elga Labwater, LLC) was used for the water flooding to obtain and to maintain initial water saturation. Paraffin oil (CAS Number 8012-95-1, Sigma-Aldrich Co. LLC) with viscosity of 130 mPa·s at 25°C and density of 868 kg/m^3, measured by the mass flow meter (M13, Bronkhorst Cori-Tech BV), was used for oil saturation processes. The *ex-situ* emulsion used in this study was oil-in-water (70.75

wt% oil and 29.25 wt% water with average oil drop size of 2.46 micron. The stable emulsion was made of heavy-oil, which is distinctly different than the resident oil (paraffin oil) in the sand pack. Density and viscosity of the emulsion are 1014 kg/m^3 and 495 mPa·s at 100 s^{-1} shear rate at 25°C (Patrick Brunelle (Quadrise Canada Corp.), 2012), respectively. Since the chosen emulsion is heavier than water and oil, it can be hypothesised that the downstream mass flow meter can detect each phase separately. By setting reference points for density of each phase in the developed software and using on-line measurements from the mass flow meter, fluid fraction-tracking software can be developed. Hence, the system can be further modified to eliminate the need for test tubes.

For each experiment, the sand (dry water-wet silica sand from Ottawa IL sand deposit with US sieve size# 10 (100-140), 53-251 μm) was weighted prior to the packing inside the core holder. The mass of the sand used for each pack was recorded to be 557 gm and was consistent from experiment to experiment with 1.5 % of deviation. The procedure for packing the core was as follows. The sand with a known mass was packed inside the void volume of the core holder (324 cm^3) using a vibration table (VP-181, FMC Technologies) with the vibrator controller (Syntron Power Pulse AC, FMC Technologies) set at maximum level. The core holder with one opened end, containing the inner plastic tube, was placed vertically on the vibration table. The sand was poured inside the core holder continuously up to the top of the internal tube. The core holder was kept for 5 minutes on a powered table to let the sand settle and form a dense medium. A rod with diameter equal to the internal diameter of the plastic tube was used to compress the sand to produce a dense pack.

Table 3.2: Properties of the sand packs

Pack	Porosity, % by water/ by mass measurements	Pore volume , cm^3	Absolute permeability, Darcy	Permeability with residual water, Darcy	Residual water saturation after oil saturation S_{wi}
Run #1	37/35	113.4	10.2	6.25	0.02
Run #2	36/36	116.0	19.8	11.2	0.18
Run #3	37/35.8	115.9	11.3	6.30	0.06
Run #4	35.8/33.4	108.1	11.3	7.90	0.06

Each pack was characterized for porosity and permeability, which were obtained prior to each flooding experiment. The obtained data are shown in Table 3.2. In order to obtain a porosity value (pore volume) for each pack, a vacuum pump (117, Labconco Corp.) was connected to the inlet side and a vacuum gage was connected to the outlet side of the core holder. The core holder was evacuated to 4 kPa absolute (28.74 in. mercury gauge or 1.18 in. mercury absolute) over half of an hour. Then the vacuum pump was disconnected and a tank filled with known volume of water, V_{kv}, was connected to the core holder and allowed to flow of water into the core holder. The remaining volume, V_{rem}, of water in the tank was recorded using a double metric scale glass cylinder (250 ml Brand 3025, Pyrex). From these measurements, the volume of water occupied by the pore space can be calculated using a modified Eq. 2.1, which provide the value of the porosity of the sand pack:

$$\phi = \frac{V_{kv} - V_{rem}}{V} \tag{3.1}$$

In addition, the porosity was calculated using the mass of packed sand, void volume inside the core holder and bulk density of silica sand (Systems, 2014) (2.65 gm/cm^3). Pore volumes (PV), the volume of fluid within the packed core holder, were in the range from 113.4 to 120.0 cm^3.

For each pack, the permeability was obtained using three different methods: with nitrogen, water and finally, with residual water saturation during oil flooding.

The absolute permeability was first found using nitrogen immediately after dry sand was packed in a core holder. The mass flow controller with built-in pressure sensor was connected to the inlet and a pressure sensor was installed at the outlet of the core holder. At different flow rates of nitrogen, the pressure difference was recorded. The absolute permeability was found using the Klinkenberg equation (Klinkenberg, 1941):

$$K_g = K_l \left(1 + \frac{b}{P_m}\right) \tag{3.2}$$

where, K_g is the gas permeability, mD; K_l is the liquid permeability, mD; b is the Klinkenberg factor, Pa; and P_m is the mean absolute pressure, Pa.

The absolute permeability with water as the working fluid was calculated for water saturated pack using Eq. 2.11. Prior to obtaining permeability measurements with residual water saturation, the oil saturation process was carried out using paraffin oil. The oil was injected into the water saturated pack with a constant injection rate of 0.5 cm^3/min for 1 PV of injection; thereafter, the injection rate was increased to 1.0 cm^3/min for an extra 0.5 PV of injection, until the residual water saturation, S_{wi}, was reached (see Table 3.2). At the end of the process, the flow rate was reduced back to the 0.5 cm^3/min and the permeability was calculated using Eq. 2.11. The effluent stream was collected in batches in test tubes. The volume of water and oil collected were compared to the injected one and used to obtain values for residual water saturation, shown in Table 3.2. Porosity and permeability measurements showed that sand packing process was consistent for all one dimensional experiments experiments.

3.5 Experimental Validation of the Core Flooding System

3.5.1 Effluent Management System

The comparison of different flooding strategies in terms of oil recovery depends on the underlying physical processes in terms of the displacement of the resident oil phase by the invaded flooding agent. At the pore-level flooding strategies can be evaluated using a Reservoir-on-a-Chip (ROC) (Gunda et al., 2011) type system. However, on a macro-scale, where integrated cumulative measurements take place, it is important to estimate the volumes of respective phases through a reliable effluent management system. Thus, it is important to calibrate and ensure that the downstream stream is measuring both the volume of water and oil for the effluent streams in developed system. In developed core flooding system, two different approaches for the downstream analysis were utilized. The fractions collected into the test tubes with fraction collector were analyzed using mass flow meter data at first and later through visual inspection of test tubes.

In order to record the volume of each phases, the effluent flow through the mass flow meter was analyzed. A density reference point, ρ_{ref}, was set at 935 kg/m^3 for measurements during the four runs, corresponding to different flooding experiments. This reference point was selected based on initial calibration runs with paraffin oil and DI water. As the liquid flows through the downstream mass flow meter during a single time step, t_{step} ($\approx$ 1 second), the density, ρ_{record} (kg/m^3) and the current volume flow rate Q_{cur} (cm^3/min) were recorded. Based on these quan-

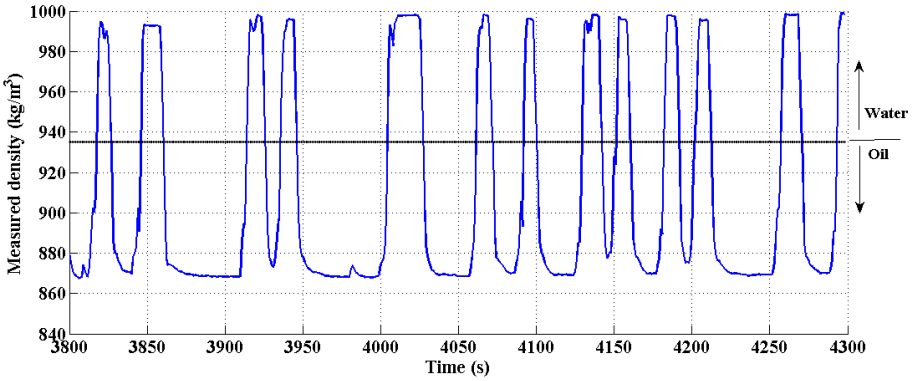


Figure 3.10: An example of density measurements at the effluent side during core flooding experiment.

tities, the fluid volume for a given phase,$V_{j,rec}$ (cm^3), was calculated as:

$$V_{j,rec} = \frac{Q_{cur} * t_{step}}{60}; \;\; j \;\; \text{= oil or water} \tag{3.3}$$

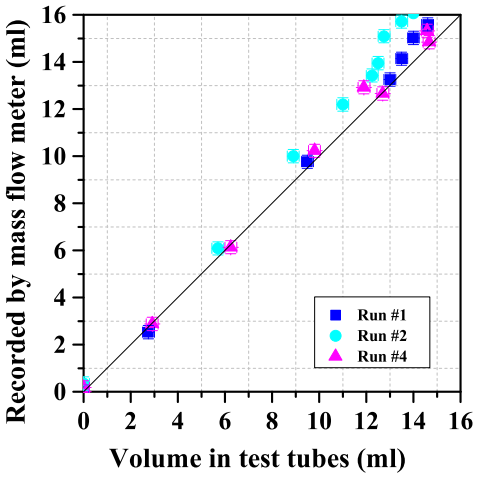
An example of density measurements at the effluent side is shown in Figure 3.10. If the density was larger than the reference level ρ_{ref}, the recorded volume (Eq. 3.3) was added to the water value $V_{water,rec}$ and the oil value $V_{oil,rec}$ was kept unchanged. Otherwise, it was vice versa. This approach was also used for emulsion flooding experiments, where the values above the reference level represented water and emulsion phases.

In total four runs - two water flooding experiments (Run #1 and Run #2), direct emulsion flooding (Run #3), and finally water flooding followed by emulsion emulsion flooding (Run #4) have been performed. For this purpose, the fractions of the effluent were collected into the test tubes and compared to the recorded volumes from the mass flow meter data and are shown in Figures 3.11 and 3.12. The error bars correspond to the error in mass flow meter and fraction measurements (the complete error analysis is available in Appendix A-2). The vertical axis here shows the total volume of each liquid recorded by the mass flow meter and the hor-

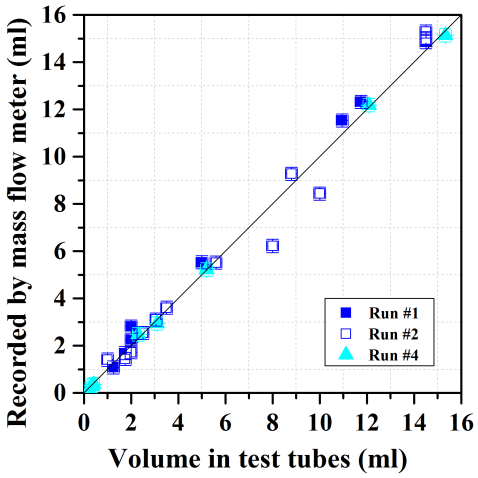
izontal axis represents the total volume collected in test tubes. Each test tube could be either filled with single liquid or can have multiple liquids, e.g. oil and water. Consequently, points on the graphs could be in a range from 0 to 15 ml. The diagonal line in these figures provides the measure of the data accuracy between two measurement techniques.

Figure 3.11 contains data points obtained during the process of water flooding for different runs, including one where the emulsion flooding was followed after the water flooding step (Run #4). Data points for produced water, shown in Figure 3.11 (a), follow the 45° line trend. Figure 3.11 (b) shows the similar trends for the recovered oil in the effluent stream. This highlights that measurements from the mass flow meter can be used for online determination of the effluent fraction.

Results obtained for the two emulsion flooding methods are presented in Figure 3.12. In Figure 3.12 (a) the results of produced mixture of emulsion and water are depicted and Figure 3.12 (b) shows the volume of recovered oil measured by the two techniques. Even though it can be observed a strong correlation between the two measurement techniques for certain measurement range, this is not true for the entire range of measured data, as was the case for Figure 3.11. In both cases, data correlation between the two measured techniques is not as strong compared to the water flooding results shown in Figure 3.11. This suggests further calibration for different regimes are required to increase the accuracy level(due to the limited amount of available emulsion, this has not been done in this research study). The worst deviation from the 45° line trend was found the be $\pm$ 22 %.

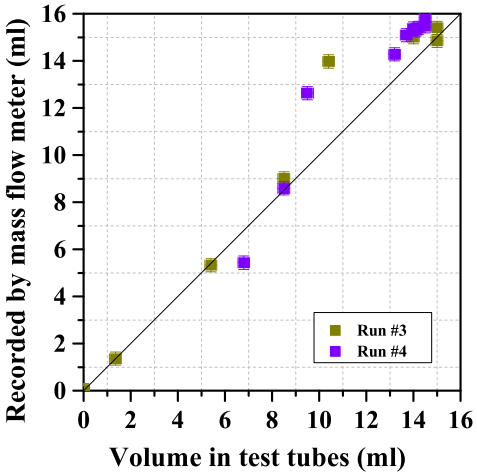


(a) Produced water

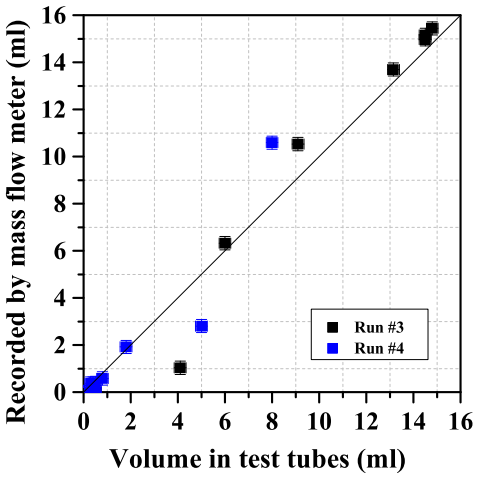


(b) Recovered oil

Figure 3.11: A comparison of data obtained from mass flow meter and test tube fractionation during water flooding. The vertical axis - recovered mixture/oil recorded by mass flow meter, the horizontal axis - recovered mixture/oil measured with test tubes.



(a) Produced mixture of emulsion and water/only water



(b) Recovered oil.

Figure 3.12: A comparison of data obtained from mass flow meter and test tube fractionation during emulsion flooding. The vertical axis - recovered mixture/oil recorded by mass flow meter, the horizontal axis - recovered mixture/oil measured with test tubes.

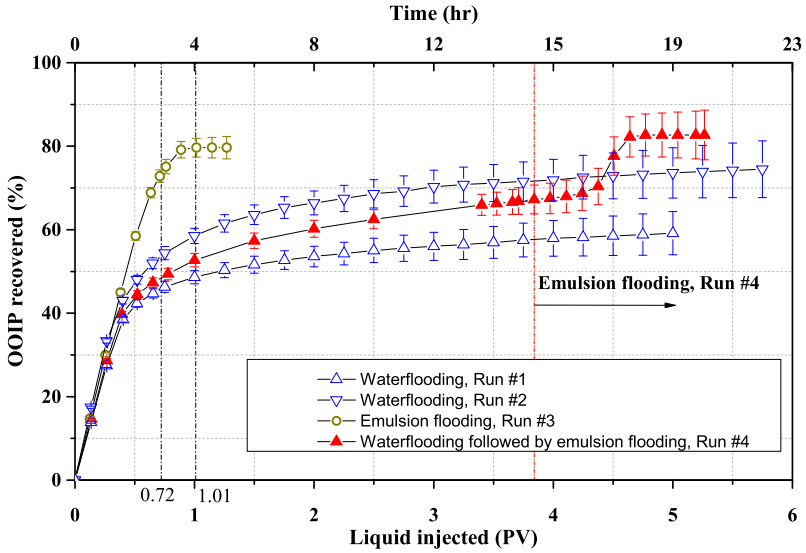


Figure 3.13: Recovery comparison for studied cases: Run #1 and Run #2 - water flooding, Run #3 - direct emulsion flooding and Run #4 - water flooding followed by emulsion flooding.

3.5.2 Core Flooding Experiments

Once the effluent management system is properly characterized and tested, the core flooding system was used to perform different flooding experiments. 19 to 22.5 hours are required to complete single experiment at a selected flow rate. Having reliable software with a graphical user interface allowed to run system overnight. The recovery curves for each set of flooding experiment is shown in Figure 3.13. Each recovery curve represents the cumulative oil production during single run/experiment. The presented data was obtained using determined volumes from test tube fractionation which were matched to the data collected from the effluent mass flow meter. Points used to plot recovery curves are collected from the test tubes analysis. The upstream and downstream mass flow meters supplemented mass balance calculations. It can be observed that the first two water flooding runs ended with recovery around 60% and 75% for Run #1 and Run #2, respectively. The water appeared at

the effluent side (breakthrough point) for Runs #1 and #2 occurred at 0.3 and 0.24 PV of injected displacement fluid, respectively. The obtained numbers were similar to the published data for unconsolidated sand packs with porosity around 30-40 % and permeability from 1.1 to 3.4 Darcy (Hadia et al., 2007b; Mandal et al., 2010b; Hadia et al., 2008a). It was observed that after 2 PV of injection of water, the ultimate recovery increased by only 6% and 8% for Run #1 and Run #2, respectively. Loose sand pack in Run #2 with higher absolute permeability compared to Run #1 resulted in higher oil recovery during water flooding. The data suggests that the pore volumes used in the experiments and the corresponding increase in the recovery are similar to the field applications (Dake, 1998, p.364). Alternate injection strategy must be incorporated to have economic recovery rate. Hence, instead of continuing with water flooding beyond a certain PV, emulsion was introduced to the system at 3.7 PV in Run # 4.

For Run #3, the emulsion was used as a primary flooding agent for the oil saturated porous media. As observed from Figure 3.13, the emulsion flooding alone produced larger oil recovery compared to water flooding for the same PV of injection. During the emulsion flooding almost 60% OOIP was recovered after injecting only 0.5 PV of emulsion. The total oil recovery was 79% OOIP at 1.25 PV of injection. From 1.01 PV of injection, additional oil recovery was not observed for Run #3. From the effluent analysis it was identified, that the breakthrough for this case happened at 0.72 PV of emulsion injection. Such a high efficiency of the emulsion flooding compared to the water flooding could be due to delayed breakthrough and having its viscosity similar to the viscosity of the paraffin oil. Emulsion saturated front moves steadily through porous media due to low viscous ratio between the

displaced and the displacing fluids (Doorwar and Mohanty, 2011).

For Run # 4, the complete water flooding experiment was carried out up to 3.7 PV of water injection. The recovery curve follows a similar trend to the Run #1 and Run #2, as is shown in Figure 3.13. At 3.7 PV of water injection, emulsion was introduced to the porous media as post water flooding process to enhance oil recovery. The total oil recovery was found to have increased. An additional 15 % of OOIP was recovered. The emulsion injection beyond 4.6 PV did not displace additional oil from the sand pack. Based on effluent collected into test tubes at the effluent side, similar to the Run #3, the emulsion saturated front moves steadily through a porous media. Complete understanding for the emulsion flooding process can be obtained by ceasing experiments and analyzing sand packs afterward. The residual oil saturation at the end of the Run #4 is around 18 %, which is similar to that in case of Run #3 with emulsion flooding only. It is to be noted that even though emulsion floodings (both Run #3 and Run #4) produced additional oil recovery ($\sim$ 20% OOIP), it required an amount of emulsion close to single pore volume to achieve such high recovery.

3.6 Conclusion

A core flooding experimental setup has been constructed to improve on the existing system which has been in use for several decades. The developed setup allows to test enhanced oil recovery techniques in more efficient time manner. The system was equipped with the up-to-date techniques and hardware available in fluid measurement systems. An automated effluent analysis incorporating properties (mass flow rate and density) measured by mass flow meters has been developed. This

allowed the replacement of the traditional fraction collector in the downstream section for online determination of effluent composition. The entire experimental setup is also monitored and controlled through a custom built software.

The core flooding experiments with unconsolidated sand packs as a medium has been performed for validation purpose. To understand the applicability of the system, emulsion (oil-in-water) used as a flooding agent, as an example of advanced technique for enhanced oil recovery. Here, two different strategies for emulsion flooding have been demonstrated - direct injection of emulsion (Run #3) and water flooding followed by emulsion flooding (Run #4). It has been shown that the engineered emulsion worked effectively as an independent flooding fluid and Run #3 produced 79% OOIP recovery at 1.2 PV of injection. Also, emulsion worked effectively as a substitute fluid after the water flooding in Run #4 and added an extra 15% OOIP to the total oil recovery at the cost of an additional 0.93 PV of emulsion injection. Comparison between these two runs suggests that, Run #3 was effective in terms of the time needed for the oil to be recovered from the porous media, while Run #4 was effective in terms of the emulsion used for the oil recovery process.

These core flooding experiments demonstrate that this newly developed core flooding system is capable of performing both the traditional water flooding and more advanced enhanced oil recovery process using *ex-situ* obtained emulsion as a flooding agent. This system can be used for academic research purpose using different types of core holders with various combinations of injection and production wells and at the same time can be adopted an industry bench mark to perform routine core studies and has a high potential of replacing the existing core flooding systems.

Chapter 4

Water-Alternate-Emulsion (WAE): A new technique for enhanced oil recovery[1]

4.1 Introduction

Existing literature suggests, that the oil and gas industry has faced a challenge working with produced water for EOR, where the goal has been to minimize the overall water injection inside the oil reservoirs. Processes were formulated where water/steam and another immiscible fluid were injected simultaneously to reduce the overall water footprint for oil recovery. Good examples of such processes are cyclic steam simulation (CSS) and water-alternate-gas (WAG) flooding (Alvarado and Manrique, 2010; Shah et al., 2010; Luo et al., 2013; Huang and Holm, 1988). In the case of CSS, the cyclic process requires around a month to implement and involves three stages: steam injection, soaking of the steam into the heavy oil formations and oil production by natural flow or by artificial lift (Shah et al., 2010; Scott, 2013). This cycle can be repeated till the oil production rate becomes uneco-

[1] A version of this chapter has been published. A. Baldygin, D. S. Nobes, and S. K. Mitra, "Water-Alternate-Emulsion (WAE): A new technique for enhanced oil recovery",*Journal of Petroleum Science and Engineering*, DOI:10.1016/j.petrol.2014.06.021, 2014

nomical (Shah et al., 2010). For WAG, different ratios for water to steam slugs for each cycle have been investigated, *e.g.* 2:1, 3:1, 4:1 and even 10:1 (Shah et al., 2010; Luo et al., 2013; Huang and Holm, 1988; Bagci and Tuzunoglu, 1998; Tüzünoglu and Bagci, 2000; Christensen et al., 2013). An increase in production was recorded up to 37% or even higher during cycling different slugs in a laboratory coreflooding system (Tortike, 1991). In order to further improve the WAG flooding technique an alternative technology was developed referred as chemical-augmented WAG (CAG), which replaced steam slugs with water-miscible chemical (alkali/surfactant/polymer) slugs (Luo et al., 2013).

All these mentioned techniques, i.e., CSS, WAG, CAG, etc., are referred as enhanced oil recovery for processes. CSS can achieve 60 % higher recovery of original bitumen in place (OBIP) compared to the steam assisted gravity drainage (SAGD) in the Clearwater formation (Scott, 2013). CO_2 - CAG allows to recover 71.07 % OOIP compared to CO_2 - WAG with 54.25 % OOIP at the same experimental conditions (Luo et al., 2013). WAG field experience shows an increase in oil recovery from 0.7 to 37 % of OOIP (Christensen et al., 2013). However, replicating these enhanced oil recovery processes with emulsion and water slugs, particularly when such emulsion is produced *ex-situ* from hydrocarbon sources quite different from the *in-situ* resident oil phase within the pores, remains to be explored. Only one study has been reported on sequential emulsion flooding (Guillen et al., 2012), where the amount of emulsion (1.65 PV) and water used (≈22 PV) was economically ineffective in terms of recovered oil and the overall time required to complete the process (Guillen et al., 2012). Hence, in this work, the focus is on using emulsion slugs alternatively with the water slugs in a cyclic manner similar to earlier

strategies for enhanced oil recovery with water and steam (CSS,WAG) or water and chemicals (CAG). Experimental results in the previous Chapter also suggested finding an alternative injection strategy to improve emulsion flooding efficiency. In this study, the emulsion flooding technique is named as water-alternate-emulsion (WAE) process.

Like any other new method for oil recovery, the technology first needs to be tested in a laboratory core flooding system (Santosh et al., 2007; Hadia et al., 2007b; Mandal et al., 2010b; Hadia et al., 2008a, 2012) before a field operation of the new process can be embarked. Hence, following similar trends, the WAE flooding technique was investigated in laboratory scale studies which are reported here. In this study, core flooding experiments with an unconsolidated sand pack as a model porous medium are conducted. For this purpose a biaxial core holder is used and three different water-emulsion flooding ratios, viz., 2:1, 4:1 and 5:1, are tested. The oil recovered using the WAE process is compared with the traditional flooding techniques such as water flooding, emulsion flooding, and water flooding followed by emulsion flooding, presented in previous Chapter 3. Based on the results obtained from this laboratory scale study, if the WAE method appears to be a promising option, further scale-up of this technique for field application can be readily executed (Hadia et al., 2008b; Islam and Farouq Ali, 1989, 1992; Abou-Kassem and Farouq Ali, 1995; Akin et al., 2000).

4.2 Materials and Methods

The laboratory core flooding system used in this study is the one reported in previous Chapter 3. The injection flow rate of water-alternate-emulsion slugs was set

at 0.5 cm^3/min for the entire flooding studies and temperature in the experimental facilities was maintained at room temperature (around 22°C).

The porous medium for the study is unconsolidated sand (dry water-wet silica sand from Ottawa IL sand deposit: US sieve size # 10 (100-140), 53-251μm), which was packed inside a biaxial core holder for each flooding experiment (www.corelab.com, 2014a). The core holder has an internal diameter of 38.1 mm (1.50 in) and the internal length of 321.6 mm (12.66 in) and was oriented horizontally during the core flooding e xperiments. Parameters of the sand packs used for different flooding experiments are summarized in Table 4.1. Using values for porosity, ϕ, cross section of the sand packs, A, and chosen flow rate, Q, the interstitial velocity, v, can be estimated for the studied porous media. This injection rate is equivalent to interstitial velocity of $2.187 \cdot 10^{-5}$m/s, $2.071 \cdot 10^{-5}$m/s, $2.074 \cdot 10^{-5}$m/s (6.20, 5.87, and 5.88 ft/day) for Run #5, Run #6 and Run #7, respectively. It can be defined by Tiab and Donaldson (2004):

$$v = \frac{Q}{\phi A} \tag{4.1}$$

With interstitial velocities of $2.117 \cdot 10^{-5}$m/s (6 ft/day), the saturation profiles would be independent of the capillary end effect (Tiab and Donaldson, 2004). In petroleum reservoirs with conventional multiphase flow the velocities are on the order of $3.528 \cdot 10^{-6}$m/s (1 ft/day) (Willhite, 1986, p. 21).

Values for residual water saturation, S_{wi}, absolute permeability, and porosity from Table 4.1 are consistent with available data in the literature for the core flooding experiments with unconsolidated sand packs (Kumar et al., 2010; Sarma et al., 1998; Hadia et al., 2008a; Luo et al., 2013). In addition to the presented parameters, the pore size distribution was measured using an automatic pore size ana-

lyzer (PoreMaster 33 , Quantachrome Instruments) using 1.0243 gm of sand sample with porosity of 38.9 %(loose pack). The total intruded volume was found to be 0.243183 cm^3. The pore radius lies in a range from 0.003 to 109.71 μm with an average value of 12.631 μm, as it is shown on Figure 4.1. Using data from Table 4.1 with assumption that sand pack can be represented as the bundle of capillary tubes, the pore throat radius, r, can be found using the following equation Tiab and Donaldson (2004):

$$r = \left(\frac{\phi}{8k}\right)^{-0.5} \tag{4.2}$$

where, ϕ is the porosity of sand pack in % and k is permeability of sand pack in Darcy. For Run #5, #6, and Run #7 it can be found that pore radius is equal to 13.09, 12.78 and 12.49 μm with an above assumption, respectively. The obtained values lie within the same range as the once found using automatic pore size analyzer.

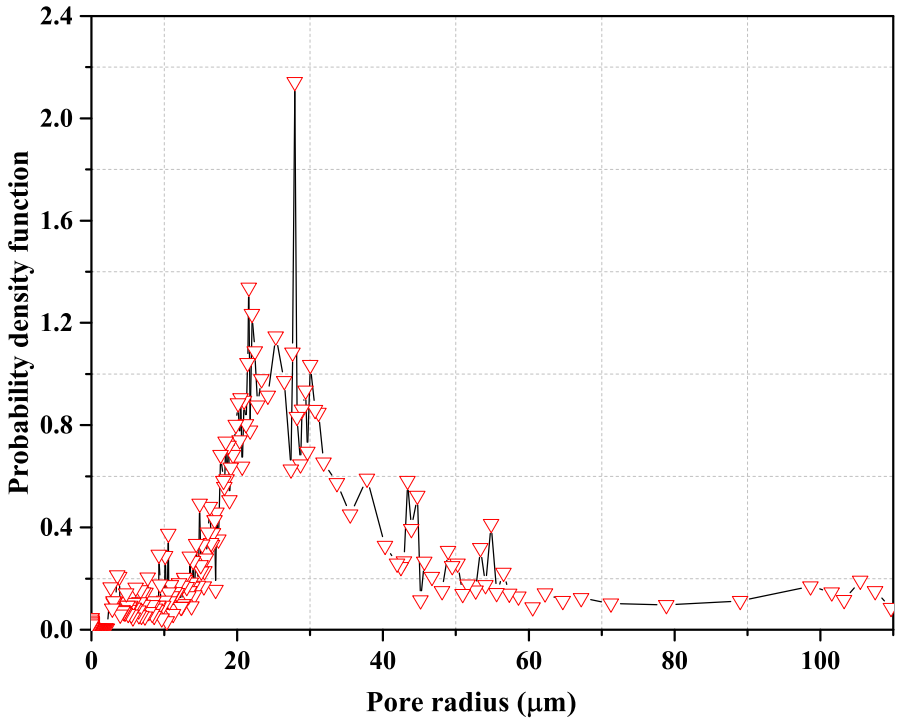


Figure 4.1: Pore size distribution of studied silica sand.

The packing process was followed the procedure discussed in previous Chapter 3. The residual water saturation was established using the following procedure. First, the sand was packed inside the core holder. The core holder was then evacuated using a vacuum pump (117, Labconco Corp.) to 4 kPa absolute (28.74 in. mercury gauge or 1.18 in. mercury absolute) over half of an hour. Graded tank was connected to the core holder and water was supplied by pressure difference to completely saturate the core. Once, the pressure of the effluent side of the core holder equals to atmospheric, the water supply was stopped and the recorded volume of water absorbed inside the core is used as the pore volume (PV), V_{pv}, for each pack. Once the absolute permeability was calculated, the paraffin oil was injected into the water saturated pack and effluent was collected into graded test tube. The injection rate was kept constant at 0.5 cm^3/min for 1 PV of injection; thereafter, it was increased to 1.0 cm^3/min for every 0.5 PV of injection till no water was observed at the effluent side. The amount of water recovered from the core holder, V_{water_rec}, was recorded. The residual water saturation, S_{wi}, was calculated with taking into account dead volumes of the system, V_d, such as supply lines, end plugs mass flow meter, etc. using:

$$S_{wi} = (V_{pv} - V_{water_rec} + V_d)/V_{pv} \tag{4.3}$$

The viscosity of the injected and produced fluids was measured using a rotational viscometer (Rheolab QC, Anton Paar USA Inc.) with a pre-installed double gap measuring system (DG42, Anton Paar USA Inc.). The density of the injected and produced fluids was measured by a mass flow meter (M13, Bronkhorst Cori-Tech BV) at the inlet and exit of the core holder. Surface tension of the injected and produced fluids was measured using the drop shape analyze (DSA 100, Krüs

GmbH) by the pendant drop technique (Waghmare and Mitra, 2010). The shear rate (100 s^{-1}) was selected as standard shear rate used in the literature (Cobos et al., 2009; Sorbie and Huang, 1991; Maia et al., 2009; Castro Dantas et al., 2006), so that it can easily compare WAE technique with previously used flooding reagents.

The internal shear rate, caused by the given injection rates used in the experiment, can be found from the known injection flow rate, sand pack dimensions, porosity and the estimated pore size. Hence, the superficial (or empty tube) velocity, U_o, can be calculated as (Darby, 2001):

$$U_o = \frac{Q}{A} = \frac{Q}{\frac{\pi d^2}{4}} = 7.309 \cdot 10^{-6}, \text{m/s} \tag{4.4}$$

Also, the velocity within the bed (i.e., interstitial velocity, U_{in},) can be calculated using porosity and the superficial velocity (Darby, 2001):

$$U_{\text{in}} = \frac{U_o}{\phi_{\text{run,n}}} \tag{4.5}$$

For Run #5, ..., Run #7 it can be found that the superficial velocity using porosity values would be equal to $2.188 \cdot 10^{-5}$ m/s, $2.071 \cdot 10^{-5}$ m/s, and $2.007 \cdot 10^{-5}$ m/s, respectively. Finally, the shear rate, γ, based on the given injection rate can be estimated using the formula for the shear rate at the inner wall of a Newtonian fluid flow inside a pipe of diameter d (or radius r) (Darby, 2001):

$$\gamma = \frac{8U_{\text{run,n}}}{d} = \frac{4U_{\text{run,n}}}{2r} \tag{4.6}$$

For Run #5, ..., Run #7 it can be found that the shear rate with selected injection rate would be equal to 6.687 s^{-1}, 6.482 s^{-1}, and 6.653 s^{-1}, respectively. Taking this into account, the viscosity presented in this Chapter of the paraffin oil, produced oil and emulsion has been measured at 6 s^{-1} in addition to 100 s^{-1}.

Table 4.1: Relevant parameters for unconsolidated sand pack samples used in flooding experiments.

Pack	WAE ratio	Porosity, %	Pore volume, cm^3	Mass of sand pack, g	Absolute permeability, Darcy	Permeability with residual water, Darcy	$S^{\ddagger}_{wi}$
Run #5	2:1	33.4	117.01	617.5	7.0 .. 7.5	5.5 .. 5.8	0.096
Run #6	4:1	35.3	123.83	599.4	7.2 .. 7.4	6.4 .. 6.8	0.085
Run #7	5:1	35.2	123.37	600.7	6.9 .. 7.0	6.8	0.085

‡ - Residual water saturation after oil saturation.

Initial water saturation and water flooding were completed using distilled water (DI) produced from a water purification system (PURELAB© Ultra, Elga Labwater, LLC). DI water used for this study has a density of 1000 kg/m^3(measured using mass flow meter (M13, Bronkhorst Cori-Tech BV)) and measured viscosity of 1.0 mPa·s at 500 s^{-1} shear rate and surface tension of 72.40 mN/m. Oil saturation was executed using paraffin oil (CAS Number 8012-95-1, obtained from Sigma-Aldrich Co. LCC) with a viscosity of 122.7 mPa·s and 120.5 mPa·s at 6 s^{-1} and 100 s^{-1} shear rate , respectively. The density of the paraffin oil was 868 kg/m^3 and surface tension of 29.85 mN/m. Stable oil-in-water emulsion (70.75 wt% oil and 29.25 wt% water with average oil drop size of 2.46 μm) was procured from a commercial provider (Quadrise Canada Corporation, Canada) (Patrick Brunelle (Quadrise Canada Corp.), 2012). It should be noted that the emulsion used for flooding was made from heavy oil which is noticeably different from the residual oil (paraffin oil). The emulsion had a viscosity of 1084.9 mPa·s at 6 s^{-1} shear rate and 495 mPa·s at 100 s^{-1} shear rate and density of 1014 kg/m^3. It was difficult to measure the surface tension of this emulsion using pendant drop method as symmetric drop generation was not achievable for a fluid of such high viscosity.

Three different ratios of water to emulsion slugs, 2:1, 4:1 and 5:1 were tested. For these tested WAE slug ratios, the water amount was kept fixed at 0.5 pore

volume (PV). Hence, the emulsion PV was varied accordingly to maintain the ratios between the water to emulsion slugs. As an example, 4:1 WAE slug would mean water with 0.5 PV and emulsion with $0.125(= 0.5/4)$ PV. In the case of 2:1 slug (Run #5), the WAE method has been applied only for two cycles, which amounts to cumulative injection of 1 PV of water and 0.5 PV of emulsion. In the case of 4:1 (Run #6) and 5:1 (Run #7) slug ratios, the WAE method was applied for three cycles, which amounts to 1.5 PV of water and 0.375 PV of emulsion for 4:1 ratio and 1.5 PV of water and 0.3 PV of emulsion for 5:1 ratio, respectively. It should be noted that at the end of each cyclic WAE process, only water injection has been performed, the amount of PV injected for each run is different. In addition, the effluent analysis has been performed to compare the viscosity and surface tension between the injected and the produced fluids.

4.3 Results and Discussion

The aim of this study was to demonstrate the efficiency of an enhanced oil recovery technique, viz., WAE flooding, using laboratory core flooding experiments. In this section, detailed results for WAE are provided and this technique is compared with traditional water flooding or with emulsion flooding alone. These provide a comparative indicator of the efficiency of the proposed WAE technique. First, a qualitative analysis was performed through actual visualization of the sand pack after the WAE flooding and later, quantitative data in terms of pressure profiles during water-alternate-emulsion injection and resultant oil recovered from the sand pack during WAE flooding was provided.

4.3.1 Qualitative analysis of WAE Flooding

Figure 4.2(a) shows the cross-section image of the actual sand pack corresponding to Run #5 (i.e., WAE ratio 2:1), which was removed from the core holder at the end of Run #5. This includes two cycles of WAE followed by $\approx$4.1 PV of water flooding. Visual inspection of the core pack suggests that there is a large build-up of the heavy oil component of the emulsion at the inlet section of the core. The hypothesis is that after the first cycle of WAE, further water injection for the next cycle causes water to interact with the emulsion available within the pore space of the sand pack. This destabilizes the emulsion resulting in phase segregation and thereby more of the heavy oil fraction of the emulsion is accumulated at the inlet section and the water phase of the emulsion is transported to the downstream section of the core.

(a)

(b)

Figure 4.2: Cross-section images of the sand pack (a) after complete WAE flooding with 2:1 ratio followed by water flooding up to 5.6 PV; (b) after complete water flooding followed by approximately 0.3 PV of emulsion injection. Here, the direction of injection is from left to right.

In contrast, when first traditional water flooding (≈3.5 PV of injection) followed by only emulsion injection (≈0.3 PV of injection) was performed, it was noticed that there is a piston like displacement of the resident oil and water phases by the injected emulsion. This is evident by a clear vertical front movement of the emulsion, as observed in Figure 4.2(b). Such piston-like displacement has been reported earlier for emulsified solvent flooding (Sarma et al., 1998; Shah et al., 2010; Mahmood and Brigham, 1987).

The amount of emulsion used in WAE flooding was both 0.52 PV and 0.3 PV for water flooding followed by emulsion flooding. The emulsion front progressed through the sand pack up to 9 cm in both cases. It indicates that for the alternative flooding, the emulsion stays inside the pack and slowly builds up the emulsion front at the inlet region cycle after cycle.

4.3.2 Quantitative Analysis of WAE Flooding

Pressure drop was continuously monitored and measured by differential (dP1 in Figure 3.1) and two inline pressure sensors(TP-2 and TP-1 in Figure 3.1) (FP2000 series, Honeywell International Inc.) across the core holder for each WAE ratio. Figure 4.3, 4.4, and 4.5 shows the pressure profiles for different WAE ratios along with the corresponding oil recovery curve. The raw pressure data obtained from the measurement has been processed using B-spline interpolation using graphical and data analysis software (Origin 9.1, OriginLab, Co.) to remove noise and smooth data, and the obtained values are plotted here.

In Figure 4.3, there is the graph where the left vertical axis is a percentage of original oil in place recovered from a porous media, the right vertical axis is a

pressure drop measured across the core holder and the horizontal axis is a volume of liquid injected normalized by a pore volume. The horizontal dashed lines indicate the location of the pressure jumps. There are two inflection points in the pressure drop curve which correspond to emulsion injection during the cyclic WAE process. During the first cycle, the introduction of the emulsion phase resulted in a gradual increase of pressure inside the core till the end of the emulsion injection with a resultant in total differential pressure rise of 12 psi. This pressure rise data is also confirmed by the visual inspection of the sand pack (Figure 4.2(a)) where most of the emulsion is plugged inside the pore-space which may result in the increase in overall pressure drop. This process has also been observed by others and is often referred as a blockage process (Decker and Flock, 1988). Also, the effluent analysis for this cycle suggested that the oil recovered from the core pack corresponds to paraffin oil only without any traces of the heavy oil phase of the emulsion. This again emphasizes that most of the emulsion is trapped at the inlet of the core pack as observed through visual inspection.

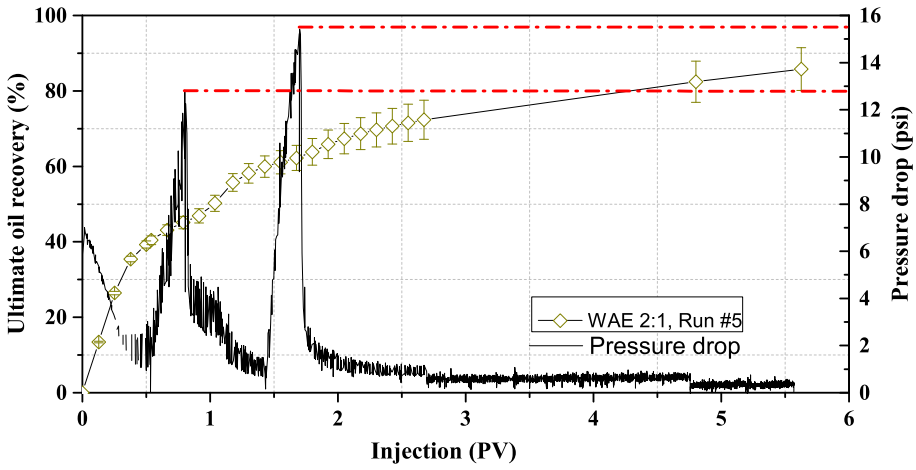


Figure 4.3: Variation of pressure drop across the core holder and the oil recovery with respect to PV of injected fluids, 2:1 WAE. The horizontal dashed lines indicate the location of the pressure jumps.

After the end of emulsion injection shown in Figure 4.3, water injection starts which is evident from the gradual drop in pressure across the core, allowing more oil to be recovered. This additional oil recovery is observed through the sudden change in the slope of the oil recovery curve during the start of the water injection. This pattern continues till the next WAE cycle, however the jump in the oil recovery is not as much as compared to the first cycle. The reason for comparatively higher recovery for the first cycle can be attributed to the much broaden pressure drop over 0.5 PV of water injection and higher oil availability compare to the following cycles. It confirms that the given emulsion blocks high permeable zones and redirects following water flooding to the oil saturated regions. Further water flooding breaks through the new regions and results in a pressure drop across the core holder after breakthrough. The oil recovery data shows that at the end of the second WAE cycle, the oil recovery from the core pack is around 60%. After the completion of the cyclic WAE process, water injection was always conducted to recover further resident oil. In this case, such water flooding eventually results in almost 86% recovery of the resident oil with only 0.5 PV of overall emulsion injection. The oil recovery for 2:1 WAE ratio can be compared with data presented in previous Chapter 3 involving direct emulsion flooding into a similar core holder (with slightly shorter internal length (284 mm (11.18 in)). It was found that this present technique produced higher oil recovery and more importantly using significantly less emulsion compared to direct emulsion injection of 1.01 PV used in previous experiment.

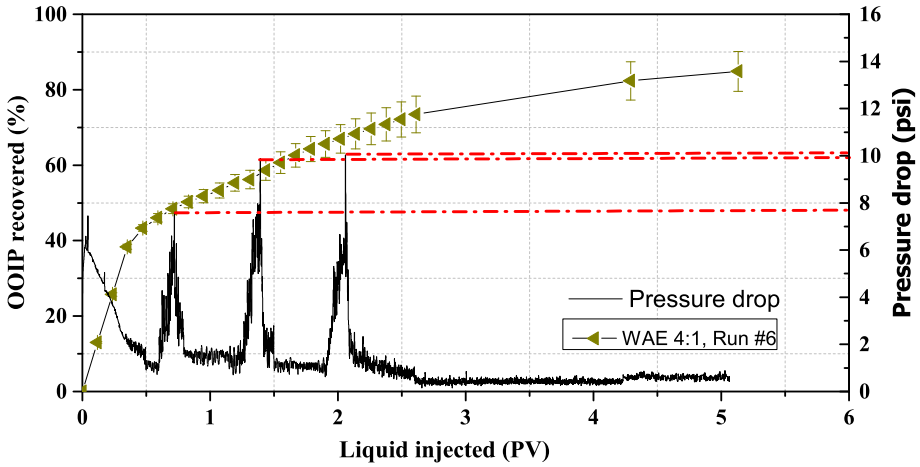


Figure 4.4: Variation of pressure drop across the core holder and the oil recovery with respect to PV of injected fluids, 4:1 WAE. The horizontal dashed lines indicate the location of the pressure jumps.

A similar trend in pressure drop was found for 4:1 WAE flooding, as shown in Figure 4.4. Since emulsion was injected three times during 4:1 WAE flooding, there are three inflection points which correspond to emulsion injection during the cyclic WAE process. During the first cycle, the introduction of the emulsion phase resulted in a gradual increase of pressure inside the core till the end of the emulsion injection with a resultant in total differential pressure rise of 8.3 psi. During both the second and the third cycle, the differential pressure across the core holder has increased to almost 10 psi which corresponds to the second and the third peaks in the pressure profile data. Also, in this case, no emulsion was observed at the effluent stream at the end of the first cycle which suggests the presence of a similar blockage process as observed for 2:1 WAE ratio has also occurred for this case. At the end of three cycle of WAE flooding, the recovered oil is ≈65% of OOIP and the final ultimate oil recovery equals to 85% of OOIP for overall 0.375 PV of emulsion injection.

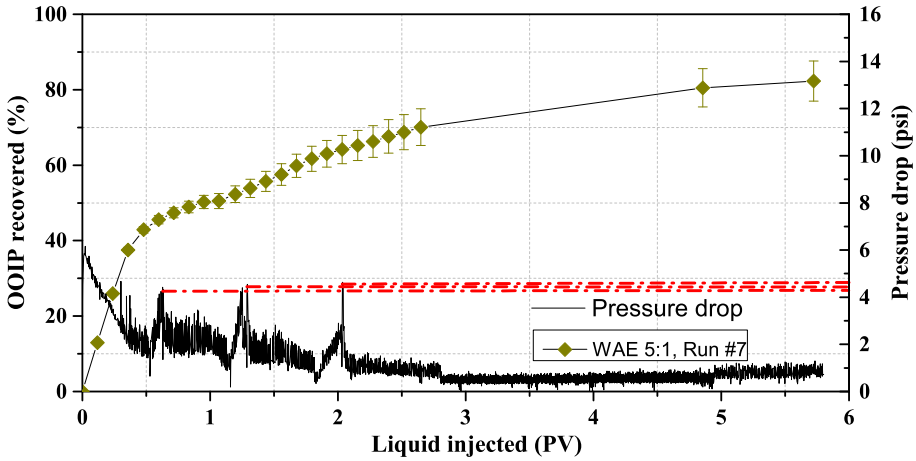


Figure 4.5: Variation of pressure drop across the core holder and the oil recovery with respect to PV of injected fluids, 5:1 WAE. The horizontal dashed lines indicate the location of the pressure jumps.

For the case of 5:1 WAE ratio, as observed for Figure 4.5, it was found that the pressure rise corresponding to each cyclic emulsion injection is almost three times lower than 2:1 WAE ratio. This can be attributed to the lower amount of emulsion injected in each cycle (0.1 PV). However, the overall oil recovery is similar to the other two ratios but with significantly less amount of injected emulsion ($\approx$0.3 PV). At the end of three cycles of WAE flooding, the recovered oil is $\approx$ 62% of OOIP and the final ultimate oil recovery equals to 82% of OOIP for overall 0.3 PV of emulsion injection.

4.3.3 Effluent Analysis[2]

Each effluent fraction, obtained from the different WAE flooding studies was analyzed to observe any change in density, viscosity and surface tension with respect to the original injected fluids. Here, the data for the 2:1 WAE flooding process is

[2]The photograph presented in this section has been published as conference paper. A. Baldygin, D. S. Nobes, and S. K. Mitra, "Oil recovery from porous media using emulsion",*The ASME 2014 International Mechanical Engineering Congress & Exposition*, IMECE2014-37248, 2014

provided and summarized in Table 4.2. A photograph of liquids injected and collected during during complete 2:1 WAE flooding is shown in Figure 4.6. Recovered emulsion is not present in a bottle, since it undergoes a separation process inside the porous media, as it was hypothesised in Section 4.3.1. It was found that the produced water, after the completion of two cyclic WAE flooding, is light brown in colour, which is quite similar to the emulsion colour. Recovered oil has dark yellow color, which is in between emulsion and produced water in a color scheme.

Figure 4.6: A photograph of liquids injected and collected during complete WAE flooding. From left to right: water, paraffin oil, emulsion, recovered oil, produced water.

The density of produced water is measured to be 970 kg/m^3 and viscosity is 1.2 mPa·s at 500 s^{-1} shear rate. Also, the surface tension of the produced water is 40.50 mN/m. It is observed that the density and the surface tension of the produced water are lower compare to the injected DI water, while its viscosity is higher. The measured density, viscosity, and surface tension of the recovered oil fraction at the effluent are 865 kg/m^3, 112.3 mPa·s at 6 s^{-1} shear rate, 112.0 mPa·s at 100 s^{-1} shear

Table 4.2: The physical properties of the liquids shown in Figure 4.6.

Liquid	Density, kg/m^3	Viscosity at 100 s^{-1}, mPa·s	Viscosity, at 6 s^{-1}, mPa·s	Surface tension, mN/m
Water	1000	1.0‡	-	68.40
Produced water	970	1.2‡	-	40.50
Paraffin oil	868	120.5	122.7	29.85
Recovered oil	865	112	112.3	29.85
Emulsion	1014	495	1085	n/d

‡ - measured at 500 s^{-1}.

rate, and 29.85 mN/m, respectively. It is found that the density and surface tension of the recovered oil is similar to the injected paraffin oil, while the viscosity is lower than the paraffin oil. These observations confirm that the emulsion blockage inside the sand pack and the surfactant present in the emulsion interacts with the injected DI water, thereby lowering its surface tension. The recovered oil has properties close to the the injected paraffin oil, which suggests that the heavy oil component of the emulsion remains trapped inside the core pack, with almost negligible amount being displaced out of the core holder.

The hypothesis stated in Section 4.3.1 is substantiated with effluent analysis, the water fraction recovered shows different surface tension and viscosity compared to the injected DI water. This may result in a change of the internal mobility ratio between the resident oil phase and the invading water phase, which may aid in recovering more oil from the core pack.

4.3.4 Comparison of WAE with Traditional Flooding

Figure 4.7 shows the cumulative recovery curves for the three different WAE ratios used in this present study compared with Chapter 3 results for water flooding, direct

emulsion flooding and water flooding followed by emulsion flooding. It is found that at least 20% more recovery can be achieved compared to traditional methods, be it water flooding or only emulsion flooding. It is also found that for a given oil recovery with any of the methods illustrated in Figure 4.7, the proposed technique of water-alternate-emulsion always requires significantly less amount of emulsion injection. For the studied parameters, it is observed that the most economical way, in terms emulsion usage, to recover oil is with 5:1 WAE ratio. In such injection strategy, only 0.3 PV of emulsion injection is required to obtain 82% of OOIP. Hence, this laboratory scale study shows promising technology breakthrough in terms of the development of an enhanced oil recovery process, which further needs to be field tested.

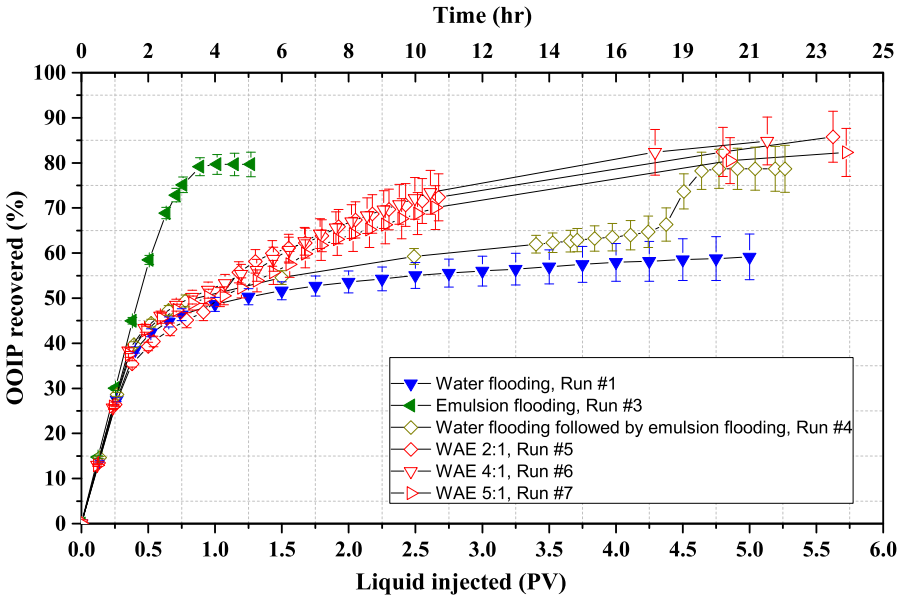


Figure 4.7: Cumulative recovery curves for WAE flooding and comparison with waterflooding, emulsion flooding, and waterflooding followed by emulsion flooding (presented in Chapter 3).

4.4 Conclusions

The present study reports an efficient technique for enhanced oil recovery with utilization of emulsion (oil-in-water) produced *ex-situ*. The developed technique, water-alternate-emulsion (WAE), was tested using laboratory core flooding system. Three different slug ratios, 2:1, 4:1 and 5:1, were experimentally investigated. The qualitative analysis of the sand packs after the WAE floodings shows that emulsion has separated inside the porous media and the heavy oil fraction of the emulsion has been trapped near the entrance region of the core. In addition from the effluent analysis, it can be assumed that the surfactant present in the emulsion combines with the injected DI water which further assists in removing the resident oil phase from the pore spaces.

The quantitative data suggests that one can achieve 22 % - 26 % more recovery with WAE compared to the traditional flooding process. In summary, the WAE flooding technique appears to be a promising enhanced oil recovery process which utilizes minimum amount of emulsion and recovers a significant portion of the resident oil. Further steps are needed to test the efficiency of this process in field reservoirs, which may have been abandoned after water flooding.

Chapter 5

Experimental flow visualization and pressure mapping of emulsion flooding with horizontal and vertical wells[1]

5.1 Introduction

The efficiency of emulsion flooding as a recovery agent (*ex-situ* produced oil-in-water emulsion) has been proved in previous Chapters 3 and 4. It was shown that emulsion flooding can potentially recover up to $\approx$ 85 % of OOIP at 0.375 PV of total emulsion injection and $\approx$ 4.5 PV of total water injection using efficient method of injection - water-alternative-emulsion (WAE) (Guillen et al., 2012; Son et al., 2014; Ponce F. et al., 2014). Other methods for injection, such as direct emulsion flooding, presented in Chapter 3 and water flooding followed by emulsion (Qiu and Mamora, 2010; Willhite et al., 1980) or surfactant/polymer flooding (Samanta et al., 2012; Kumar et al., 2012; Bryan and Kantzas, 2009; Krumrine and J.S., 1983), also showed promising results. However, the time required to recovery the same

[1] A version of this chapter has been prepared for submission to the journal. A. Baldygin, D. S. Nobes, and S. K. Mitra, "Experimental flow visualization of emulsion flooding with horizontal and vertical wells",*Journal of Petroleum Science and Engineering*, 2014

amount of oil is significantly higher. Close to 97 % of resident oil at $\approx$ 22 PV of liquids injected can be recovered using emulsion made of a synthetic oil (Shell Tivela 460) (Guillen et al., 2012).

One of the important properties of the emulsion is the drop size distribution (DSD). The conventional methods which were previously used to get the drop size distribution reported presence of single particle group (Wasan et al., 1978; Moradi et al., 2014; Romero et al., 2011; Moradi et al., 2011; Romero et al., 2006). Compared to the previously used methods used for particle analysis, currently available particle analysers count particles individually (Anderson et al., 2013). They are able to distinct different sizes and to provide total number of particles (Anderson et al., 2013). It can be hypothesised that due to the availability of particle-by-particle analyzer the presence of two distinct groups of oil particles in emulsion can be now measured.

There are two ways for emulsion interaction within porous media. Either it can increase mobility of resident oil (Mandal et al., 2010a) or block high permeable regions (McAuliffe, 1973). In previous experiments, a biaxial (Chapter 3 and 4) and Hassler core holder (Qiu and Mamora, 2010; Ko et al., 2014) have been used to simulate reservoir conditions during emulsion flooding. Even after post-experimental analysis on collected sand packs in Chapter 4, the *in-situ* interactions during emulsion flooding process are still unclear. Understanding of the emulsion flooding process at the tertiary stage without ceasing experimental progress or changing experimental parameters (repacking), its interaction with oil saturated porous media, require nondestructive testing (NDT). It is limited in the core holders previously used in full-scale studies.

The way of studying interactions within porous media without causing damage to the porous media can be by using multi-dimensional experimental setups. At the micro-scale, the converging-diverging quartz capillary tube can be used to understand the physics of emulsion flow at the scale of the pore size (Cobos et al., 2009; Romero et al., 2011). At the macro-scale, reservoir on a chip (ROC) with glass beads (Guillen et al., 2012) or artificially constructed porous media (Mei et al., 2012) can be used to mimic porous media stricture, which is preset in the reservoir. Experimental data collected from these experiments allows the validation and development of fundamental theory about emulsion flooding. Thus, the efficiency of emulsion flooding at given reservoir conditions can be predicted. However, the pore volume utilized in these studies are about 0.822 cm^3 (Mei et al., 2012) or 2.980 cm^3 (Guillen et al., 2012) which is even less than in biaxial and Hassler core holder. A step further in terms of the experimental scale can be by utilizing a two dimensional (2D) core holder with the pore network 10 times larger than in conventional biaxial or Hassler core holders.

A two dimensional core holder shows how the front of the *in-situ* flow spreads through the porous media (Santosh et al., 2007; Doorwar and Mohanty, 2011; Robert et al., 2006; Chaudhari et al., 2011b; Naderi and Babadagli, 2011). The 2D core holder represents a sectional cut of a real reservoir with dimensions obtained from scaling parameters (Hadia et al., 2008b; Islam and Farouq Ali, 1989). The side wall of the 2D core holder can be transparent or solid piece with pressure/temperature ports (Butler and Chung, 1988; Li and Mamora, 2010). It is thin enough to assume that the flow is distributed in two directions only and there is no influence (flow) in the third direction. Another aspects, such as well configura-

tion, can be studied in addition to the flood front studies. Vertical injection - top or bottom horizontal production (VI-THP and VI-BHP), horizontal injection - vertical production (HI-VP), horizontal injection - horizontal production (HI-HP), vertical injection - vertical production (VI-VP) wells configurations can experimentally investigated (Santosh et al., 2007, 2008; Hadia et al., 2008a,b).

Flow visualization and pressure mapping of the emulsion flooding process can help to understand the internal dynamics happening inside the porous media. It also provides answers as to how well a configuration can affect on efficiency of emulsion flooding or how dynamics of emulsion flooding front through previously abandoned regions depends on well configuration and what is the increase in ultimate oil production. The previously utilized experimental setup has been modified accordingly to provide answers. In this Chapter, the results of water flooding followed by emulsion flooding experiments for three different configurations are presented. Details for the experimental setup components and materials and methods used to carry out experiments are provided. Results consist of visual image analysis, pressure data analysis, effluent analysis and internal flood front velocity analysis. Collected results provide one step further towards developing theory for the scale-up of emulsion flooding technique to the field application (Islam and Farouq Ali, 1989, 1992; Abou-Kassem and Farouq Ali, 1995; Akin et al., 2000).

5.2 Experimental Setup

A hydraulic schematic of the system developed for this study, previously described in Chapter 3, is a further modification of the concept of the core flooding system. The core block was modified compared to the previous configuration. The biax-

ial core holder has been replaced with a two dimensional (2D) core holder using the "plug-and-play" advantage of the developed system described earlier. A photograph of the final version of the experimental setup is shown in Figure 5.1. Final configuration of experimental setup prepared for the flow visualization and pressure mapping of emulsion flooding with horizontal and vertical wells is shown. It has an external light source to guarantee uniform light intensity on a core holder surface. Pressure demodulators are also added to process signals from 15 pressure sensors and to demodulate and amplify the input signals to a $\pm$ 10 Vdc full-scale output.

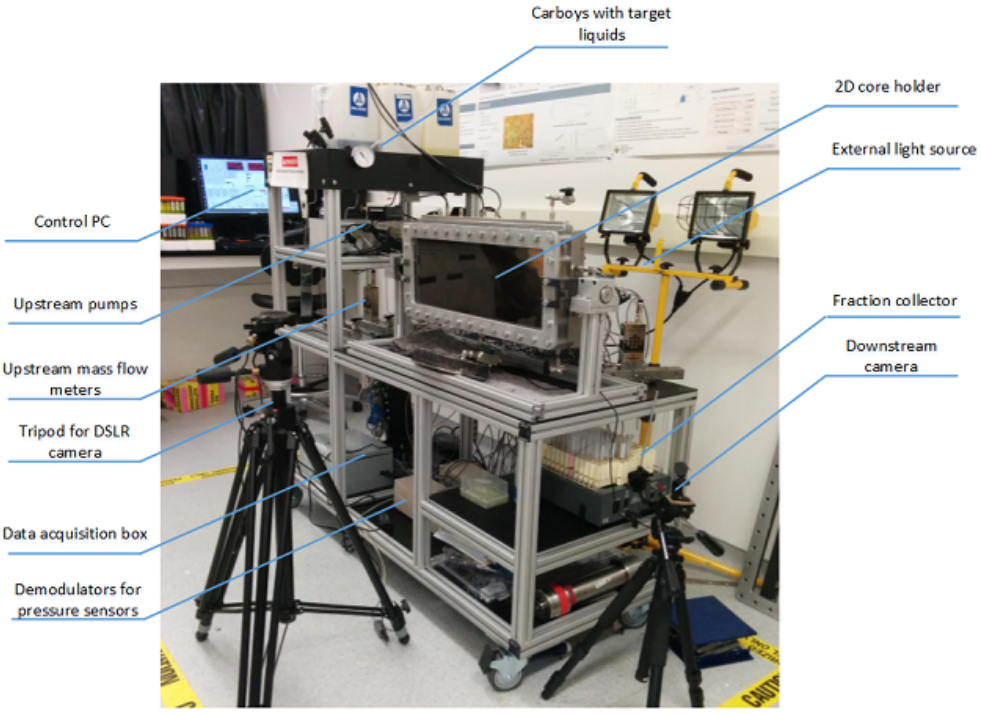


Figure 5.1: A photograph of experimental setup used for the core flooding experiments with 2D core holder.

The software with graphical user interface (GUI) used to control the system was developed based on the original version, previously discussed. Additional channels have been added to record data from extra pressure sensors installed on a two 2D core holder. Since the single experiment requires more than three days con-

tinuous running, it is crucial to provide sustained monitoring. The software was modified compared to previously released version used in previous Chapter 3 and 4 to have remote visual monitoring via a web-camera (Lifecam Studio Webcam, Microsoft, Corp.) and system monitoring via email reporting of important system parameters. The command line simple mail transfer protocol SMTP email client (sendEmail (Zehm, 2009), Brandon Zehm) was integrated into software and used for notifications. Email notifications and camera allows to monitor and to control the system 24 hours a day.

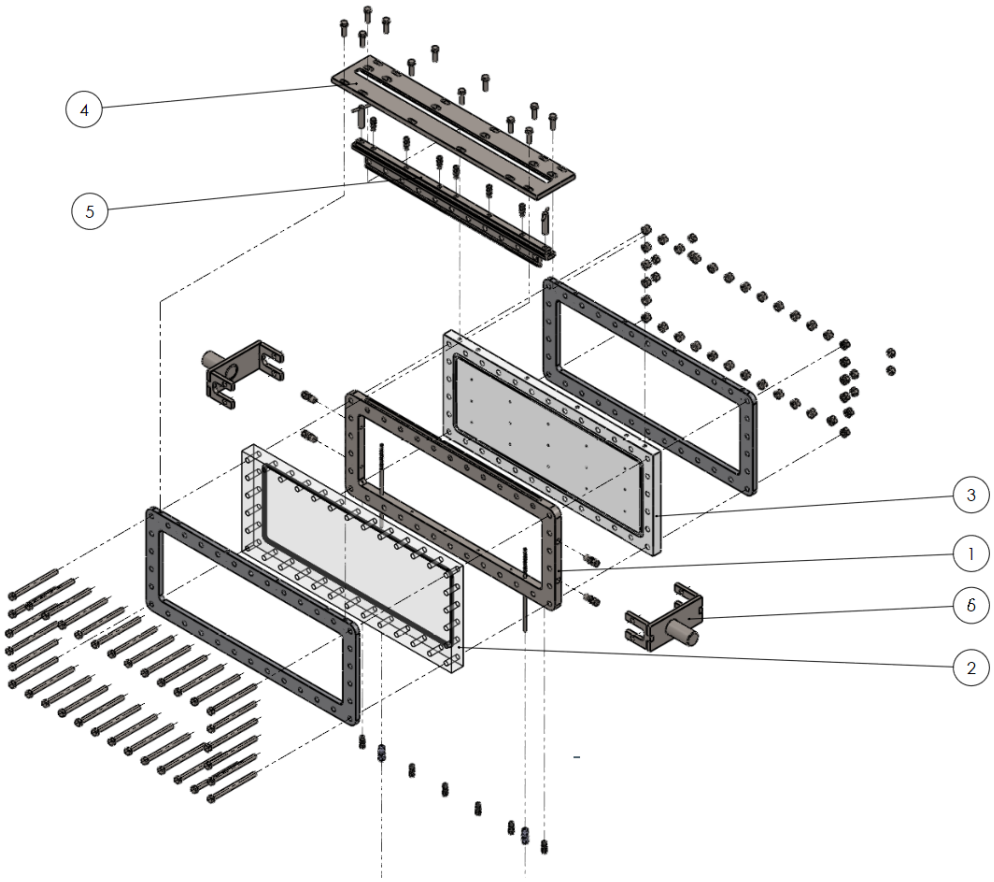


Figure 5.2: Exploded view of the 2D core holder (1) main frame with profiled slot, (2) cast acrylic wall, (3) stainless steel wall with pressure ports, (4) top cap, (5) profiled cap, and (6) holder.

The exploded view of the 2D core holder, shown in Figure 5.2, has an internal

void space of 600 x 200 x 20 mm. The main components, such as the main frame with profiled slot, cast acrylic wall, stainless steel wall with pressure ports, top cap, profiled cap, and holder are numbered. Compared to others Chaudhari et al. (2011a); Veerabhadrappa et al. (2013b), the current design has a modified configuration for the packing of sand. It has a main frame with profiled slot at the top for pouring sand inside the void space. Once the core holder is fully packed, the profiled cap closes the core holder and the top cap places on top of it. By tightening bolts, an additional compression allows to achieve a denser pack. The holder is mounted into a horizontal base (Part No. 8020-5930, 80/20 Inc.). It allows to fix at any angle and to rotate 360 degrees around horizontal axis. Well configurations can be reconfigured using multiple ports on side walls, two at the top, at the left and at the right side.

In order to recover oil from a reservoir, injection and production wells must be drilled into a reservoir. Wells used in the current study are designed in agreement with industrial requirements written for reservoirs with expected conditions close to the unconsolidated sand packs (King, 2009). Stainless steel tubes of 6.35 mm (1/4 in) outer diameter (OD) with perforations of 1.59 mm (1/16 in) were used during the core flooding experiments. The hole pattern, shown on Figure 5.3, was designed using existing recommendations (Cosad, 1992; Parrott and Walton, 2001). During the flooding process, wells were held in place with bored-through male connector (6.35 mm (1/4 in) tube OD x 1/4 in male NPT, SS-400-1-4BT, Swagelok, Co.) and were wrapped with stainless steel wire cloth (wire diameter 53.34 μm (0.0021 in.), opening size 73.66 μm (0.0029 in.), mesh size 200 x 200) (Article No. 54058, G. Bopp USA, Inc.) to provide sand control (Voll et al., 1998).

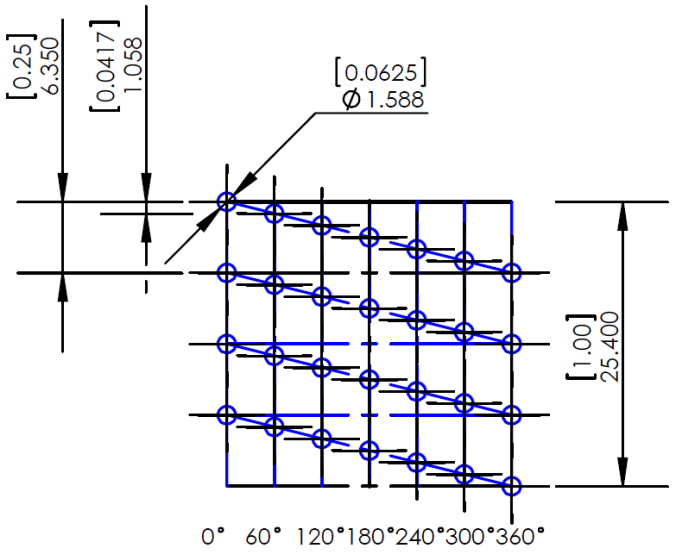
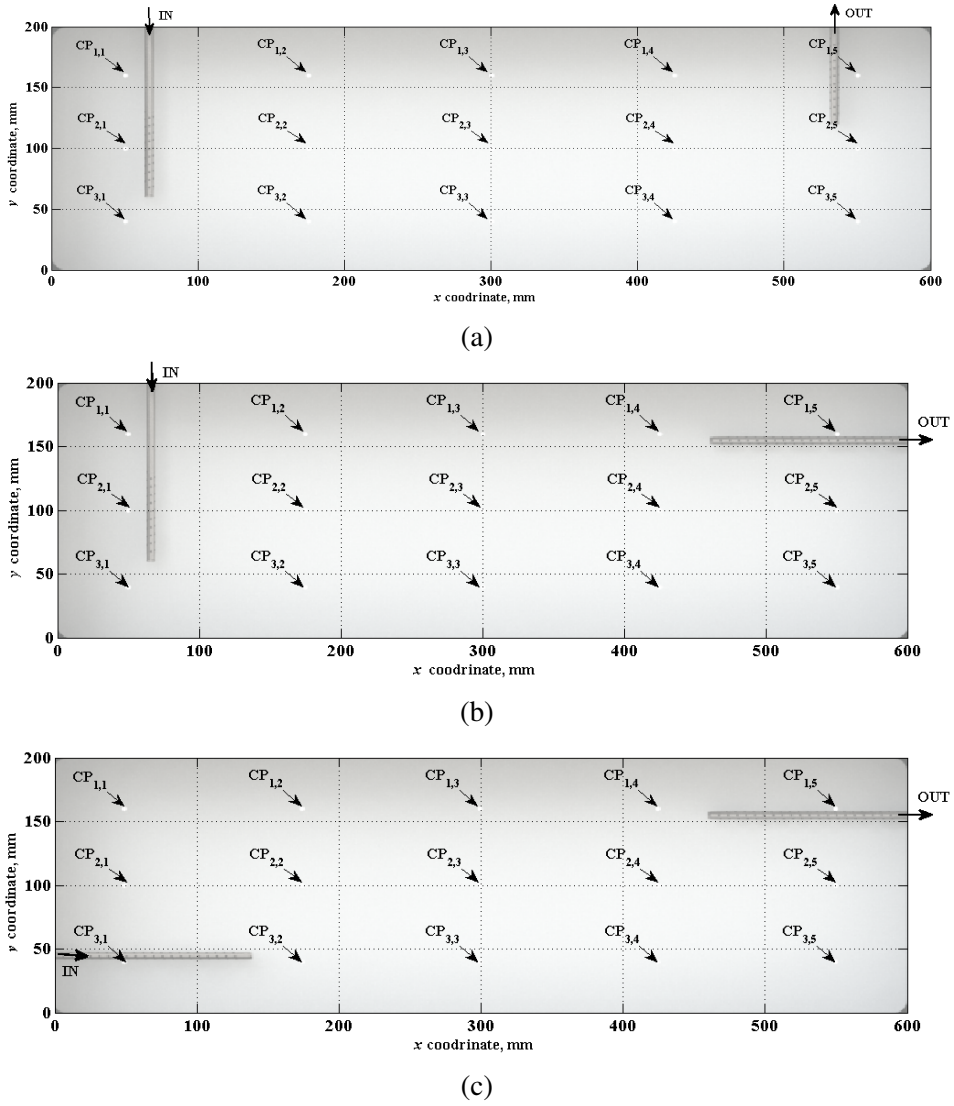


Figure 5.3: Pattern for hole perforation in wells (28 holes/in^2).

A cast acrylic sheet was installed on one side of the core holder to allow visualization of emulsion flooding with horizontal and vertical wells. A digital single-lens reflex (DSLR) camera (D7100, Nikon Canada, Inc.) with 18-105mm lens (Nikon Canada, Inc.) is mount on a tripod (Part# 028B, Manfrotto Co.) with a professional head (Part# 229, Manfrotto Co.) to capture the process of emulsion flooding and record any changes which may occur inside porous media. The images collected during flooding process have 6000 x 4000 resolution with 24 bit depth and sRGB color representation. The back plate of the 2D core holder has 15 pressure ports for monitoring the 2D pressure in the sand pack. Differential pressure sensors with replaceable membrane (DP15, Validyne Engineering Corp.) were chosen with an open end to the atmosphere. Pressure maps were generated using collected data from pressure sensors. These maps accompany data from the image analysis and provide better understanding of emulsion flooding process. Pressure sensors are named as $CP_{i,j}$, where i is the row and j is the column. Reference images with

coordinates for pressure sensors are shown in Figure 5.4. These configurations, vertical injection - vertical production (VI-VP), vertical injection - horizontal production (VI-HP), horizontal injection - horizontal production (HI-HP), were chosen as a continuation of existing experimental results with well configurations using 2D core holder for water flooding Santosh et al. (2007). It allows a comparison of results for the water flooding and benchmarking of the developed system.



(a)

(b)

(c)

Figure 5.4: Reference images with coordinates for pressure sensors in (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

The processing code for image analysis was developed using commercial multi-paradigm numerical computing environment (MATLAB R2014a, MathWorks). This code provides data for the velocity of the flood front at three vertical locations for each well configuration. The locations can be obtained by individual bases for each run, based on image resolution. The code follows the same processing steps for each run: cuts three slices for each individual image, reads out the red-channel of the image, converts the image to the binary image based on threshold, detects emulsion edge using the Prewitt method (Pre), calculates the location for the end in horizontal direction, and scales it to the actual coordinate in millimeters. Finally, it loads the obtained data and fits and plots a cubic polynomial function with robust fitting options and centers and scales (normalize) for displacement versus time. The order of a cubic polynomial function was adjusted manually from case to case to reduce numerical error. In order to obtain velocity, it finds the first derivative of the polynomial function and scales from mm/min to m/s and plots.

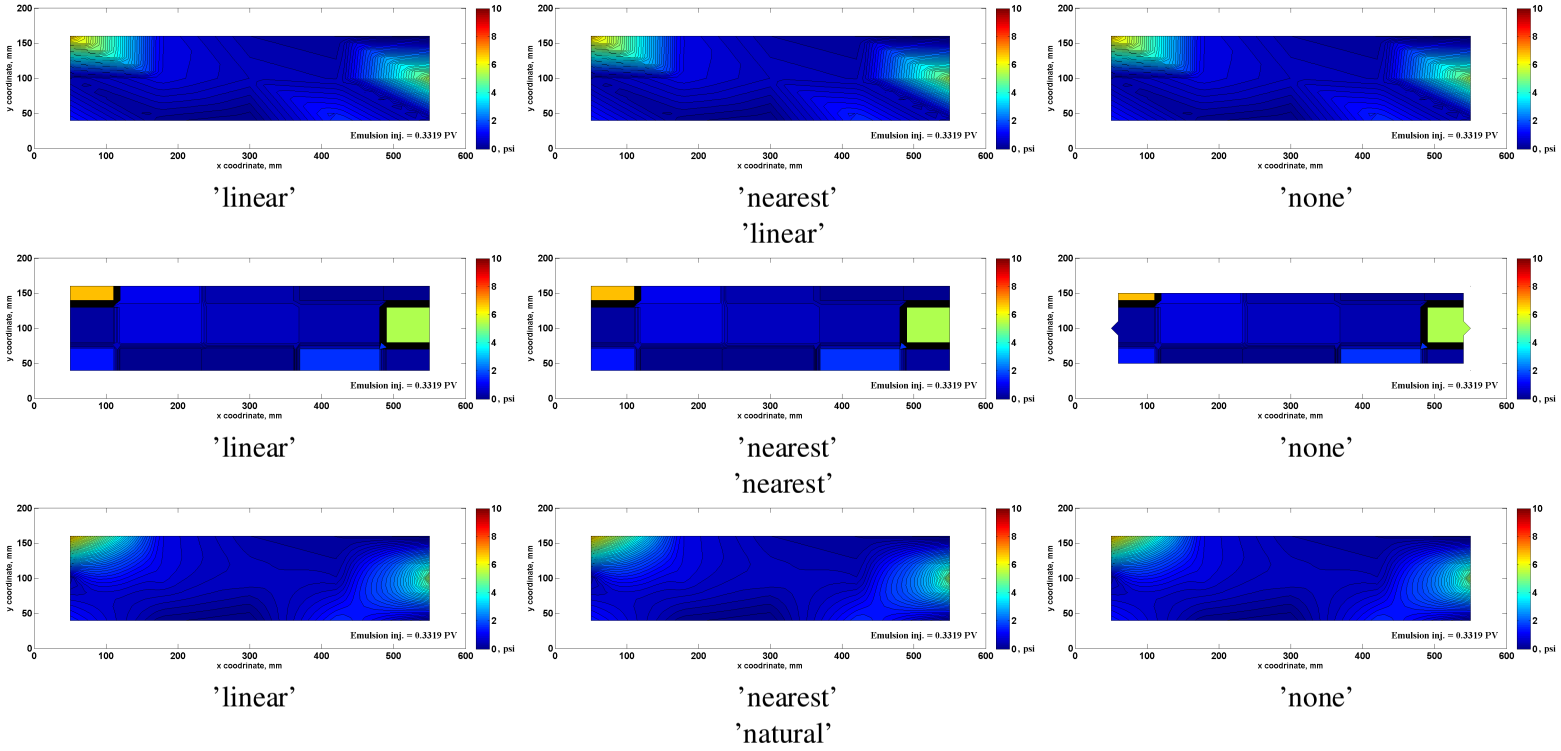


Figure 5.5: Comparison of pressure maps collected with different settings in $scatteredInterpolant$ function.

Another code was developed to analyze data from pressure sensors collected during emulsion flooding. Collected data from pressure sensors along with coordinates was prepared for plotting as filled contour plot using scatteredInterpolant function. This function allows to perform interpolation on a 2-D scattered data set.

All nine configurations for $ScatteredInterpolant$ function were investigated and are compared in Figure 5.5 to identify optimal configuration with a representation close to reality. There are three interpolation methods: linear, nearest and natural. Nearest method showed coarse results compared to two others and showed lack of representative information about *in-situ* processes. Natural neighbor interpolation re-distributes pressure equally in radial direction compared to linear interpolation with axial distribution. It is also expected that pressure follows fluid fronts. Considering images collected over different runs and presented later in this Chapter, the 'nearest' option was selected for interpolation method and 'linear' option for extrapolation method.

5.3 Materials and Methods

In this section, the details of materials and methods used to prepare and carry out experiments are provided along with a standard characterization for core flooding experiments. This data includes physical properties of utilized liquids, porosity, permeability and residual oil saturation.

The suppliers for liquids, water, oil and emulsion, were chosen the same as in previous Chapters. In order to have visible difference between liquids compared to previously reported data, the distilled water (DI) was dyed with liquid concentrate of yellow/green dye (Part # 106001, Bright Dyes, Division of Kingscote Chemicals

Inc.). Concentration was chosen as 1 ml per 1 L of DI water. The paraffin oil was dyed with liquid concentrate of red dye (Part # 506250-RF4, Bright Dyes, Division of Kingscote Chemicals Inc.). Concentration was chose as 10 ml per 5.5 L of oil. Emulsion was not dyed because of its natural high contrast compared to oil and water.

Viscosity, density and surface tension of dyed liquid were measured and compared with pure liquids to ensure that the introduced dye does not critically effect physical properties of the liquids. A rotational viscometer (Rheolab QC, Anton Paar USA Inc.) coupled with double gap measuring system (DG42, Anton Paar USA Inc.) was used to measure viscosity. A mass flow meter (M13, Bronkhorst Cori-Tech BV) was used to measure density. A drop shape analyzer (DSA 100, Krus GmbH) was used to measure the surface tension of dyed liquids using the pendant drop technique (Waghmare and Mitra, 2010). The physical properties of liquids are summarized in Table 5.1. Physical properties of dyed and pure paraffin oil have 0.67 % and 0.83 % deviation for surface tension and viscosity, respectively. Dyed water has 5.6 % deviation for surface tension. Considering difficulties associated with measuring surface tension of water, it is an acceptable deviation. Hence, both liquids can be used for the experiments without significant effect on results due to presence of dye.

The emulsion used for experiments was produced *ex-situ*. It is stable and oil-in-water type with composition of 70.75 wt % oil and 29.25 wt % water (Corp.), 2012). Quadrise Canada Inc. provided the drop size distribution which is shown in Figure 5.6. It shows normalized a Gaussian curve, which is available in the literature as was discussed earlier. A microscopic image of diluted emulsion in phos-

Table 5.1: The physical properties of the liquids.

Liquid	Density, kg/m^3	Viscosity, mPa·s	Surface tension, mN/m
Water†	1000	1.0 at 100 s^{-1}	72.40
Dyed Water	1000.4	1.0 at 500 s^{-1}	68.33
Paraffin oil†	868	120.5 at 100 s^{-1}	29.85
Dyed Paraffin oil	867	119.5 at 100 s^{-1}	29.65
Emulsion†	1014	495 at 100 s^{-1}	n/d

† - Properties for clear liquids are taken from results reported in previous Chapters

phate buffered saline pH 7.4 (PBS) solution and stabilized with surfactant (Tween 20, CAS Number 9005-64-5, obtained from Sigma-Aldrich Co. LCC) is shown in Figure 5.7. The optimal concentration was found to be 2 μL of emulsion in 49 μL of PBS with 1 μL of surfactant. Two dimensional groups can be identified from this image. First group shows the size of the drops close to 2 μm, while second group stays in a range of 10 μm. It is an evidence that the current emulsion has bimodal structure, namely two groups of particles. In order to prove the hypothesis for presence of bi-dispersal structure, emulsion has been tested using two particle analyzers (qNano and qMicro, Izon Science Ltd.). These particle analyzers use different membranes to count particles individually and provide total count per volume of liquid. Emulsion was diluted in 25 % PBS. Obtained results are presented in Figure 5.8. It provides evidence that emulsion consists of two distinct group of particles around 1.5 μm and 9 μm. The obtained curve is different compared to the supplied. Previously reported data in the literature does not provide such evidence either (Wasan et al., 1978; Romero et al., 2011; Moradi et al., 2011; Romero et al., 2006). DSD is an important parameter in fundamental theory of emulsion flooding. Having data different from existing in the literature leads to the reviewing existing

models which covers emulsion flooding process with the presence of single group of oil particles. However, it does not consider as part of the field of study in the current research project.

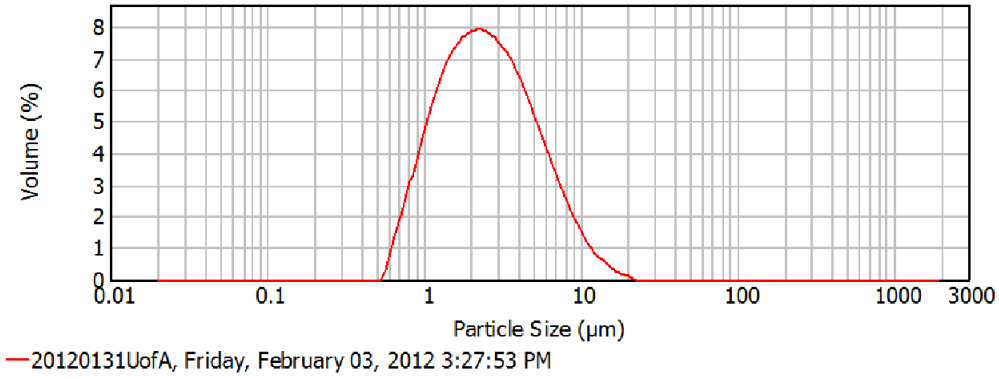


Figure 5.6: Drop size distribution measured by Quadrise Canada Inc.

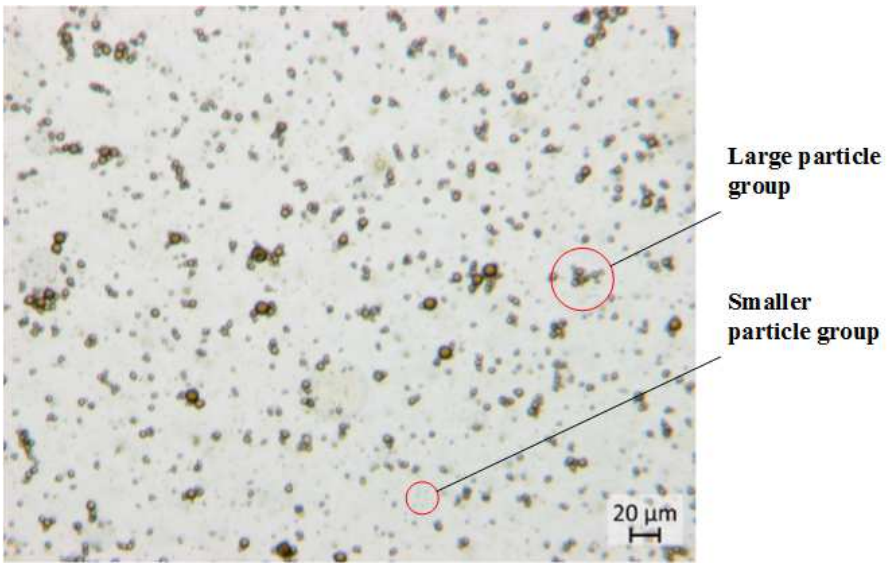


Figure 5.7: Microscopic picture of diluted oil-in-water emulsion in phosphate buffered saline (PBS), pH 7.4 (Magnification: 11.13x, Total magnification: 256.08x). The picture was taken using motorized fluorescence stereo zoom microscope (Axio Zoom.V16, Carl Zeiss AG).

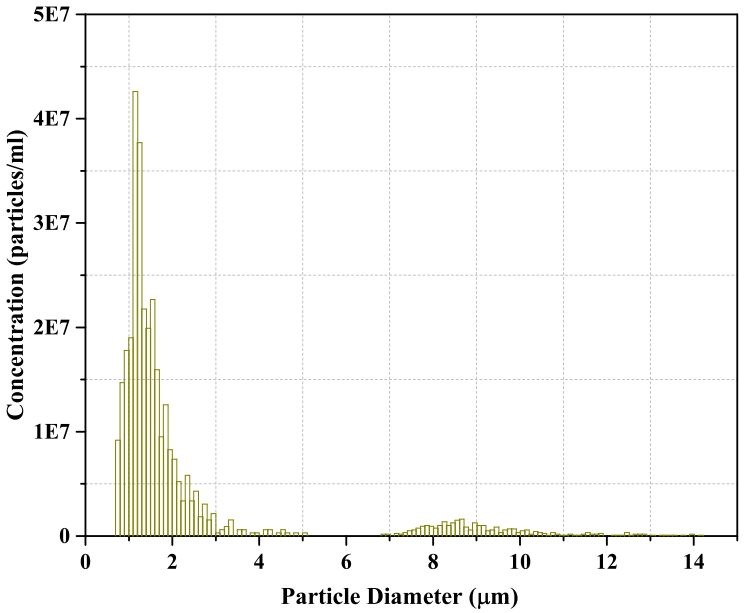


Figure 5.8: Drop size distribution measured using two particle analyzers (qNano and qMicro, Izon Science Ltd.).

The 2D core holder has been packed with unconsolidated sand from two batches. For Run #8 and Run #9, dry water-wet silica sand (Ottawa IL sand deposit, US sieve size #10 (100-140), particle size 53-251 μm) was used and for Run #10, dry water-wet testing silica sand (AGSCO Part #SSS000080-B5MBNK, US sieve size #10 (100-140), particle size 53-251 μm) was used. In order to prove that these sands can form similar porous media structure, an automatic pore size analyses (PoreMaster 33, Quantachrome Instruments) was used to measure the pore size distribution. Identical distributions were found for two batches, as it is shown in Figure 5.9. The pore size has distribution from 0.003 μm to 109.71 μm and 0.003 μm to 107.51 μm with an average 12.631 μm and 17.27 μm for silica sand and testing silica sand, respectively. Thus, sand packs would have similar pore size distributions and wetting properties.

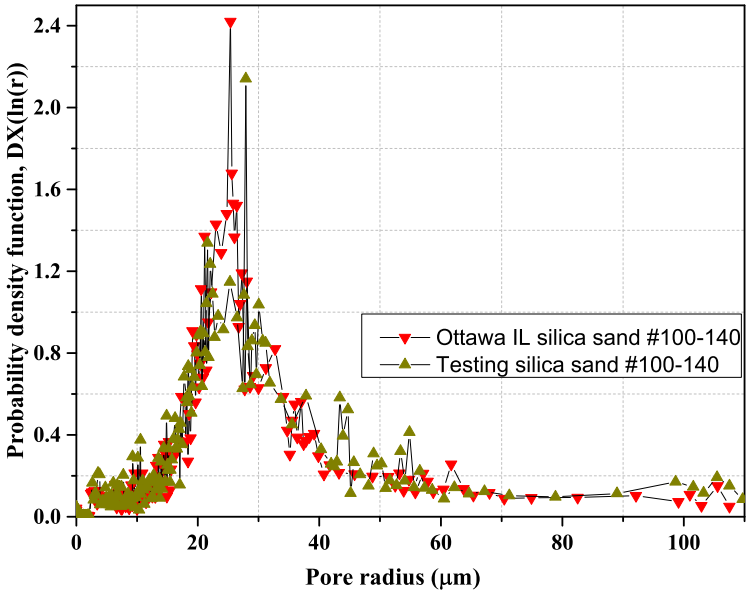


Figure 5.9: Pore size distribution of studied silica sands.

The sand packing process was different compared to previously described for biaxial core holders. The core holder was oriented vertically and fixed into the base. Dry sand with known mass was poured into the 2D core holder through the slot located at the top of the main frame. A concrete vibrator were applied on a stainless steel side of core holder for 15 minutes to create dense pack. Additional sand was added during this process, if required. At the end of this process, the slot was covered with profiled cap and compressed with top cap using bolts.

In order to saturate the core holder with water and estimate the pore volume, one side was connected to the vacuum pump (117, Labconco Corp.) and another one to a vacuum gauge. The vacuum pump was turned on for 60 minutes to guarantee vacuum level at 4 kPa absolute (28 in. mercury gauge). Once the desired vacuum level was reached, the vacuum pump was disconnected and a reservoir with a known volume of water was connected. The mass of the reservoir was measured before and

Table 5.2: Relevant parameters for unconsolidated sand pack samples used in flooding experiments.

Pack	Well configuration	Porosity, %	Pore volume, cm^3	Mass of sand pack, g	Absolute permeability, Darcy	Permeability with residual water, Darcy	$S^{\ddagger}_{wi}$
Run #8	VI-VP	35.4	847.1	4107.5	0.973 ± 0.063	0.497 ± 0.088	0.114
Run #9	VI-HP	34.6	827.7	4159.0	0.630 ± 0.083	0.248 ± 0.006	0.022
Run #10	HI-HP	34.3	820.1	4179.2	0.468 ± 0.074	0.379 ± 0.053	0.057

‡ - Residual water saturation after oil saturation.

after connection using portable balance (SPE6001, OHAUS Canada, Co.). Taking into account the density of water and the difference in volume before and after and internal dead volumes for fittings and the core holder, the pore volume was recorded.

The procedures for absolute permeability, permeability with residual water saturation and residual water saturation after oil saturation are identical to the previously discussed for biaxial core holders. The only difference in flow rates, the flow rate for oil saturation was kept at 1.75 cm^3/min and was changed to 2.00 cm^3/min at the end of oil saturation process. Relevant parameters for unconsolidated sand pack samples are summarised in Table 5.2.

5.4 Results and Discussion

The main objective of this study was to understand the behavior of emulsion flooding process in a 2D and identify the effect from well configuration on ultimate oil recovery and *in-situ* velocity profiles. The modified system has been used to carry out experiments. In this section, detailed results for visualization experiments with three well configurations, vertical injection - vertical production (VI-VP), vertical injection - horizontal production (VI-HP) and horizontal injection - horizontal production (HI-HP) have been provided along with *in-situ* the pressure mapping analysis. Since the data for the water flooding has been reported before (Santosh et al., 2007), this work focuses mainly on post water flooding process - emulsion flooding. The water flooding at 1.75 cm^3/min was performed up to 2.05 PV, 2.54 PV and 2.12 PV of water injection for VI-VP, VI-HP and HI-HP, respectively, and continued with emulsion flooding at 1.00 cm^3/min.

Image analysis was performed to compare collected photographs at different stages of experiment and understand the *in-situ* interactions. A pressure mapping analysis was used to provide support to the image analysis. Also, the effluent collected into test tubes was analyzed and effect from the configuration. The quantitative analysis is given from analysis of velocity profiles which has been compiled using processing code.

5.4.1 Visual Analysis

An example snapshot from the emulsion flooding process after water flooding is shown in Figure 5.10. The direction for the liquid flow is from the left to the right, and injection and production well was installed on the left and right side, respec-

tively. The emulsion flood front can be identified due to high contrast between emulsion, oil and water. Trapped oil is an oil saturated region which is surrounded with low permeable zones. The region preferably saturated with water is named as water saturation region. A morphologically unstable interface between emulsion and oil/water present in a porous media forms a pattern, emulsion channeling (Blackwell et al., 1959). Locations for velocity measurements and wells and over all dimensions are also shown in Figure 5.10. The dark colour represents emulsion, yellow color represents water and red color represents oil saturated regions. During sand packing process different sand patterns can be formed, as it is shown in Figure 5.10.

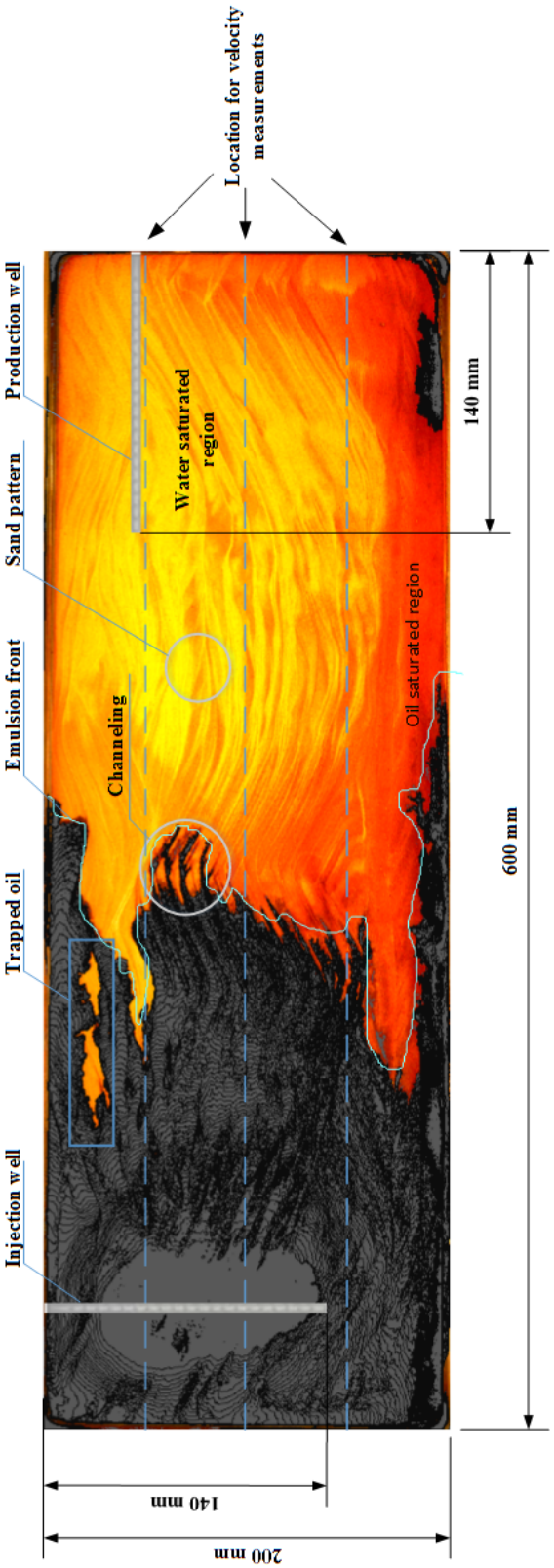


Figure 5.10: An example snapshot from the emulsion flooding process after water flooding.

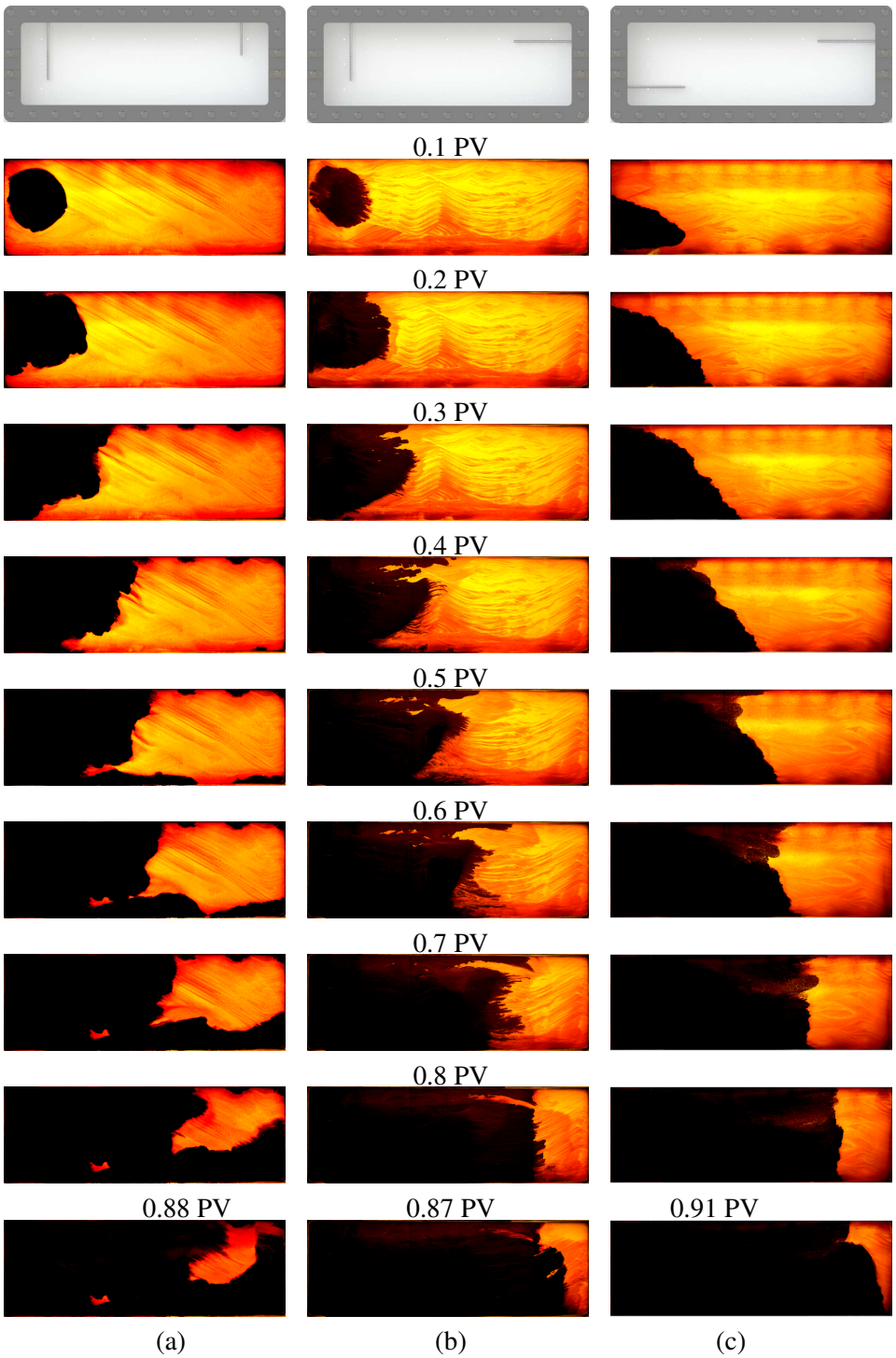


Figure 5.11: Cross-section images of the sand pack for different configurations: (a) VI-VP; (b) VI-HP; (c) HI-HP.

The photographs in Figure 5.11 (a)-(c) show snapshots of the 2D core holder at different pore volumes of emulsion injection for the studied configurations. Well configuration for each column is shown in the first image. Injection wells were half-perforated for vertical injection configurations and fully perforated for others with previously discussed perforation pattern.

In general, the pattern for the emulsion has similarities for VI-VP, VI-HP and HI-HP at the beginning of injection process, as it is shown in Figure 5.11 for 0.1, 0.2, and 0.3 PV of emulsion injection. The emulsion fills a region close to the left boundary first and then progresses towards the production well. For VI-VP, emulsion uses boundaries at first, due to slippage along the wall formed after water flooding process. It caused oil to be trapped inside the reservoir for VI-VP and VI-HP, as it is shown on last images in Figure 5.11. Emulsion pushes oil through the sand pack and forms a new oil saturated front. It can be observed compare images for 0.2 and 0.3 PV of emulsion injection.

An emulsion can block high permeable zones first during flooding process and re-direct post-flooding into high oil saturated regions. There is evidence of emulsion channeling in VI-VP and VI-HP configurations, as it is shown in Figure 5.11 (a) and (b). During water flooding water saturated channels inside the sand pattern are formed. The *in-situ* process at the scale of pore can be described as, oil drops present in the emulsion start to coalesce when the emulsion interacts with porous structure full of water and surfactant concentration drops down to a critical level. The result of this process is a significant local viscosity reduction for the emulsion. Taking into account that injection flow rate remains the same, local pressure reduction can occur in accordance with Poiseuille equation. This process can cause local

instabilities (Blackwell et al., 1959; Weitz et al., 1987) for the flow and the interface between two liquids cannot be predicted easily and channeling can form. Collected images support this hypothesis.

While HI-HP showed least oil production during water flooding, shown in Figure 5.11 (c), less high permeable zones were formed. As a results, the emulsion front has piston-like behaviour (Sarma et al., 1998; Shah et al., 2010; Mahmood and Brigham, 1987) and no channeling can be observed. Similar results were obtained during analysis of extracted sand packs from biaxail core holder in Chapter 3.

Flow visualization shows that VI-VP has a higher recovery during the water flooding process. However, emulsion flooding was not effective and caused oil to be trapped behind the emulsion front. At the same time, HI-HP was the most stable method for the emulsion injection after water flooding process.

5.4.2 Pressure Maps

Data collected from 15 pressure sensors was analyzed and plotted as filled contour plot. The filled contour plots represent the pressure map inside the 2D core holder, as it is shown in Figures 5.12 - 5.20, at given amounts of emulsion injected into porous media. From analysis of the data it was found that the maximum pressure gained during emulsion flooding inside the core holder is different from configuration to configuration. The maximum pressure drop across the core holder was found to be 9.5 psi, 59.4 psi and 114.4 psi for Run #8, Run #9 and Run #10, respectively. The color bars on the contour plots are set for three ranges 0 to 10 psi, 0 to 60 psi and 0 to 115 psi, respectively to the maximum pressure for each configuration. The difference in pressure drop scale from experiment to experiment is due to initial

conditions of porous media prior emulsion flooding. In Run #8 with VI-VP configuration, most of the oil was washed out during water flooding, even close to the boundaries. It allowed emulsion to penetrate through porous media without resistance, slipped along boundary and resulted in 10 psi pressure rise. While for Run #9 with VI-HP configuration, sand pack was still highly oil saturated, as it is shown in Figure 5.11 (b), after water flooding, emulsion at first blocked high permeable zones and proceeded with a piston-line flood front pattern. It resulted in a continuous pressure rise up to 59.4 psi at the end of emulsion flooding process. Similar was to Run #9 was found for Run #10 with even higher rise up to 114.4 psi.

The pressure during VI-VP rises after the initiation of the injection process and reaches a maximum around 0.4 PV of emulsion injection, as it is shown in Figure 5.15. After this point, pressure returns to the initial point. Comparing the pressure map with cross-section images for VI-VP, it can be seen that the emulsion slips across the boundary after 0.4 PV of emulsion injection and the drop in pressure values happened at the same time. Hence, there is *in-situ* bypassing process during VI-VP configuration.

For other configurations, VI-HP and HI-HP, pressure raises from the start of the emulsion flooding process and continues to a maximum by the end of injection, as it is shown in Figure 5.20. The direction of pressure contours has similarities with the emulsion front shown in the cross-section images. For HI-HP configuration in Figures 5.17 and 5.18, the front of emulsion goes with the same angle to the horizontal plane. Similar contours are found in the pressure maps. However, there are still differences in the data. A low pressure gap in the pressure data is observed for all configuration compared to the image data with continuous emulsion slug along

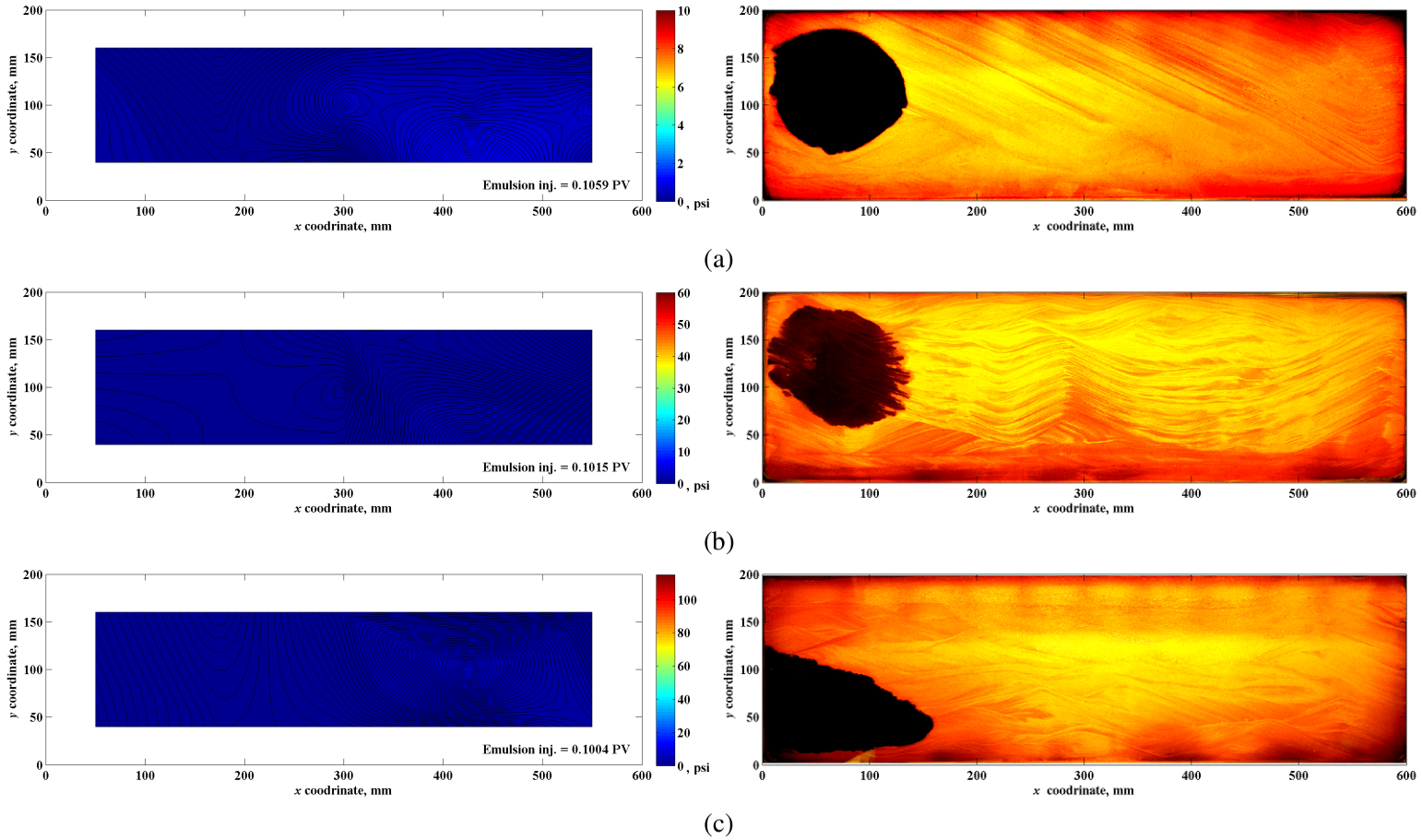


Figure 5.12: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

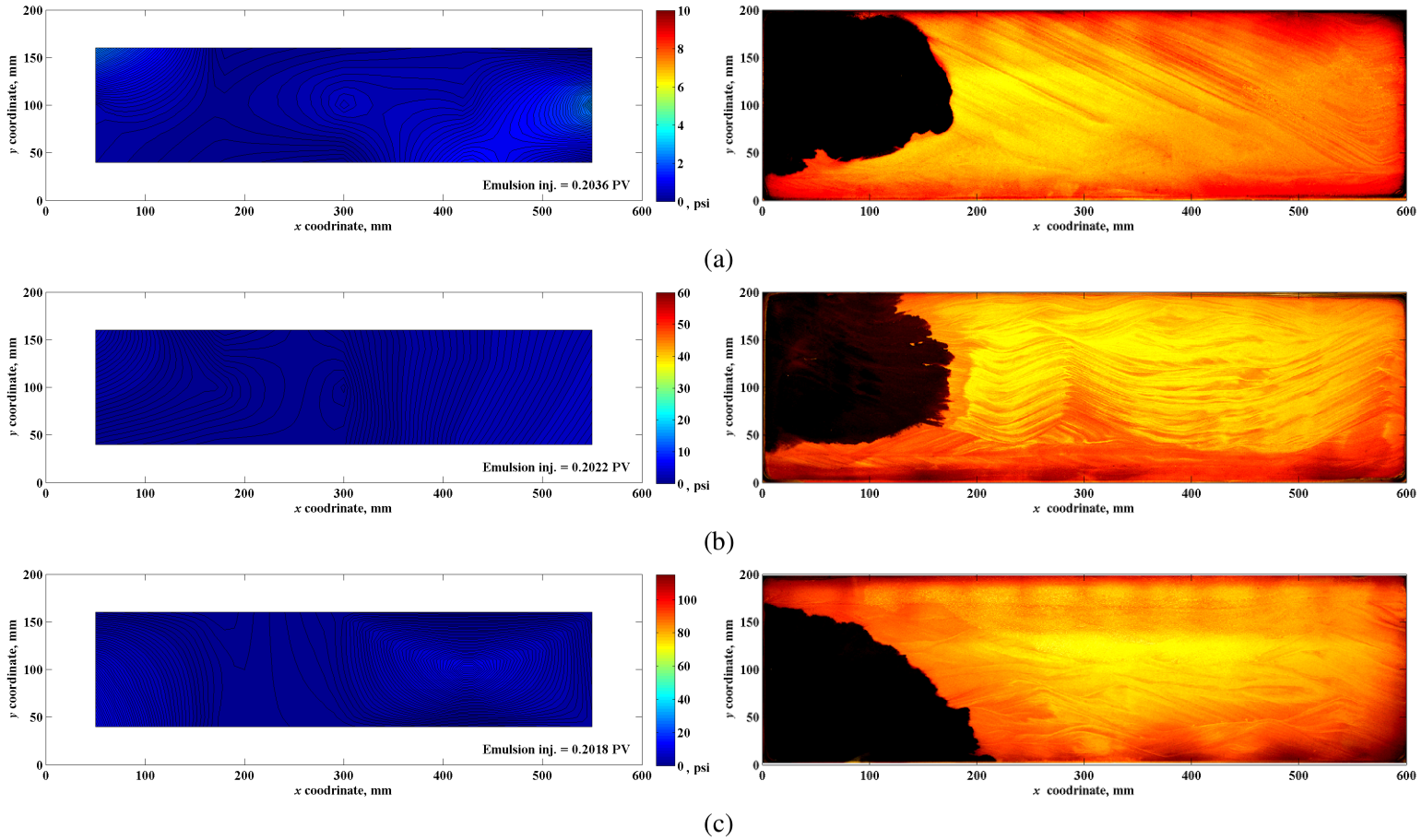


Figure 5.13: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

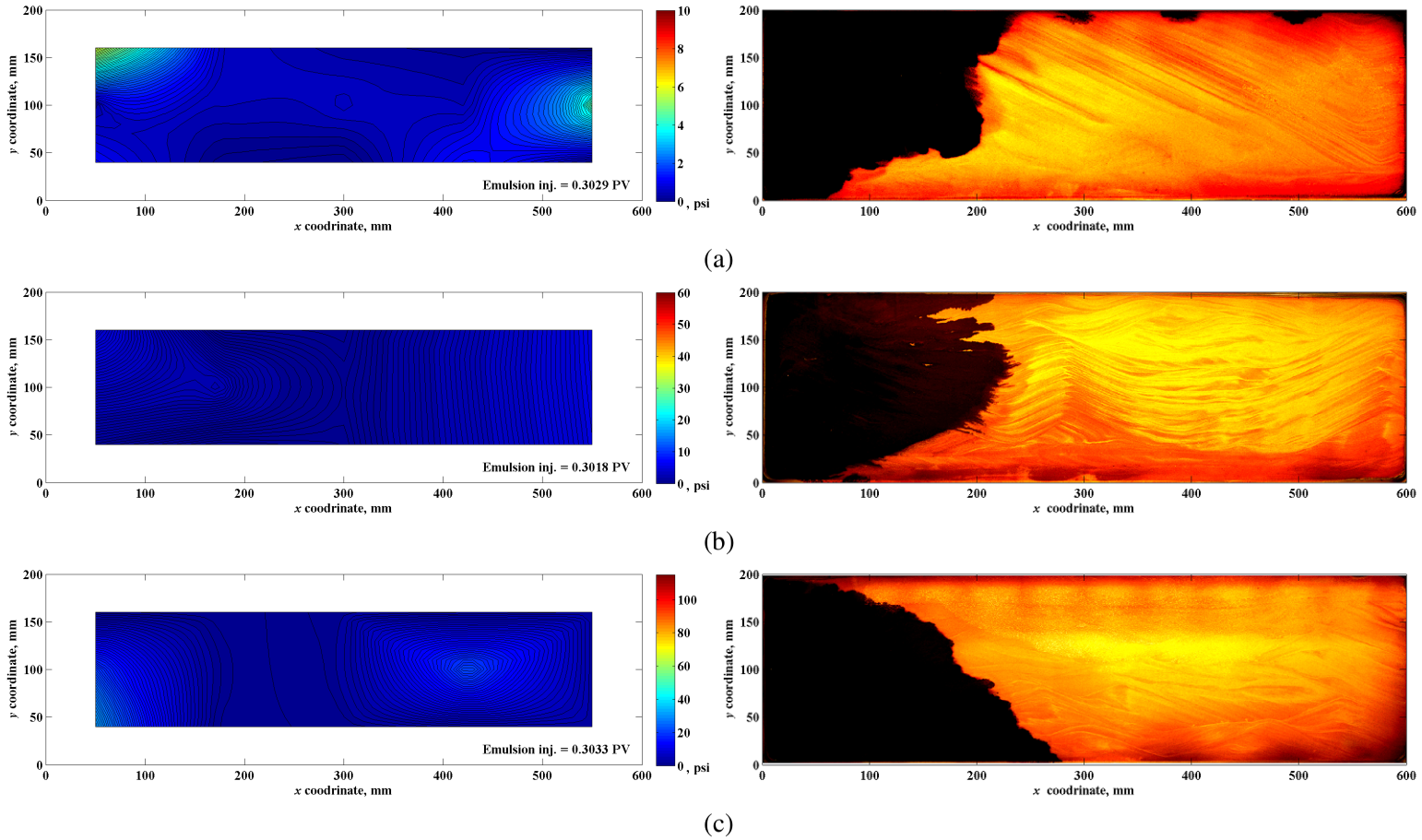


Figure 5.14: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

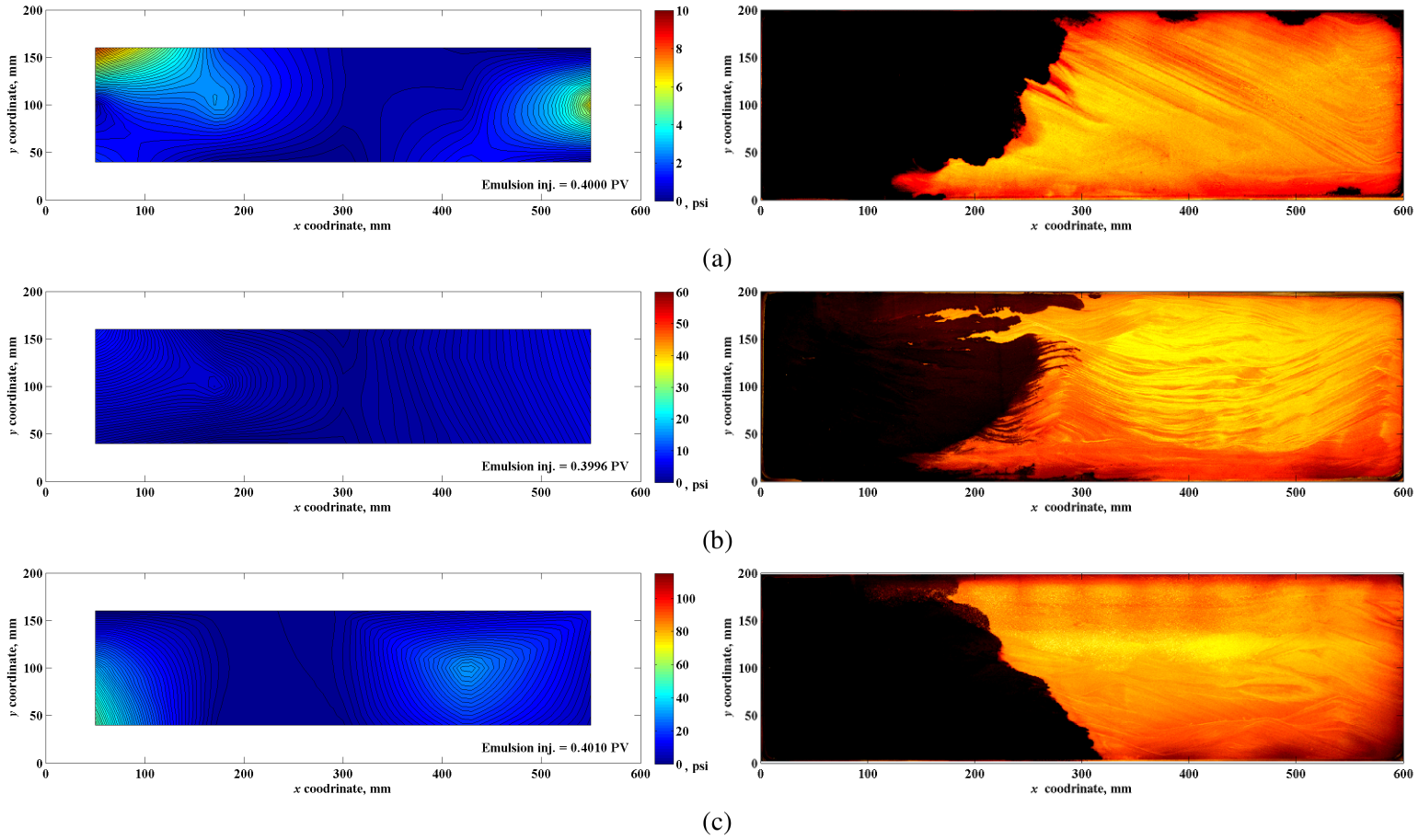


Figure 5.15: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

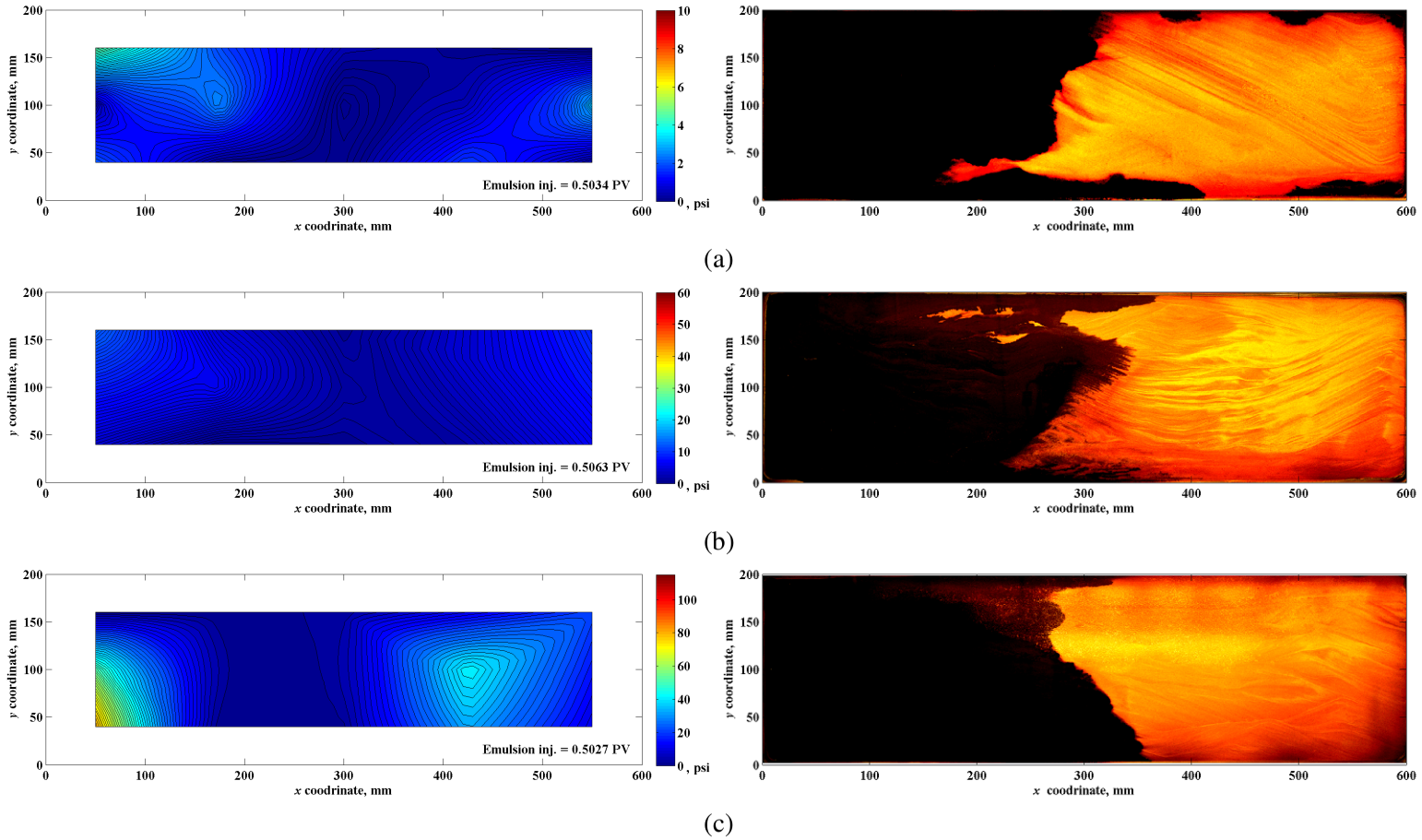


Figure 5.16: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

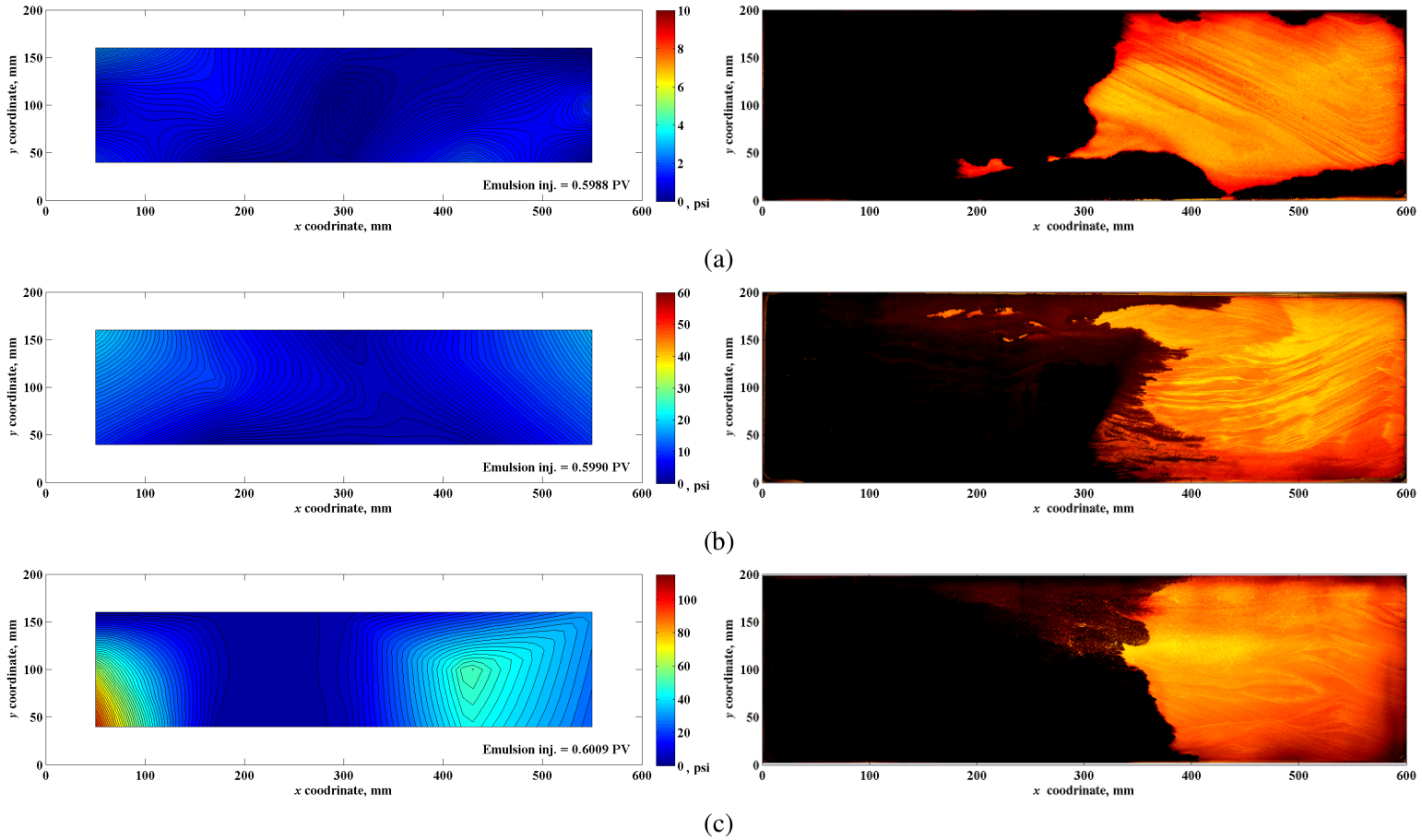


Figure 5.17: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

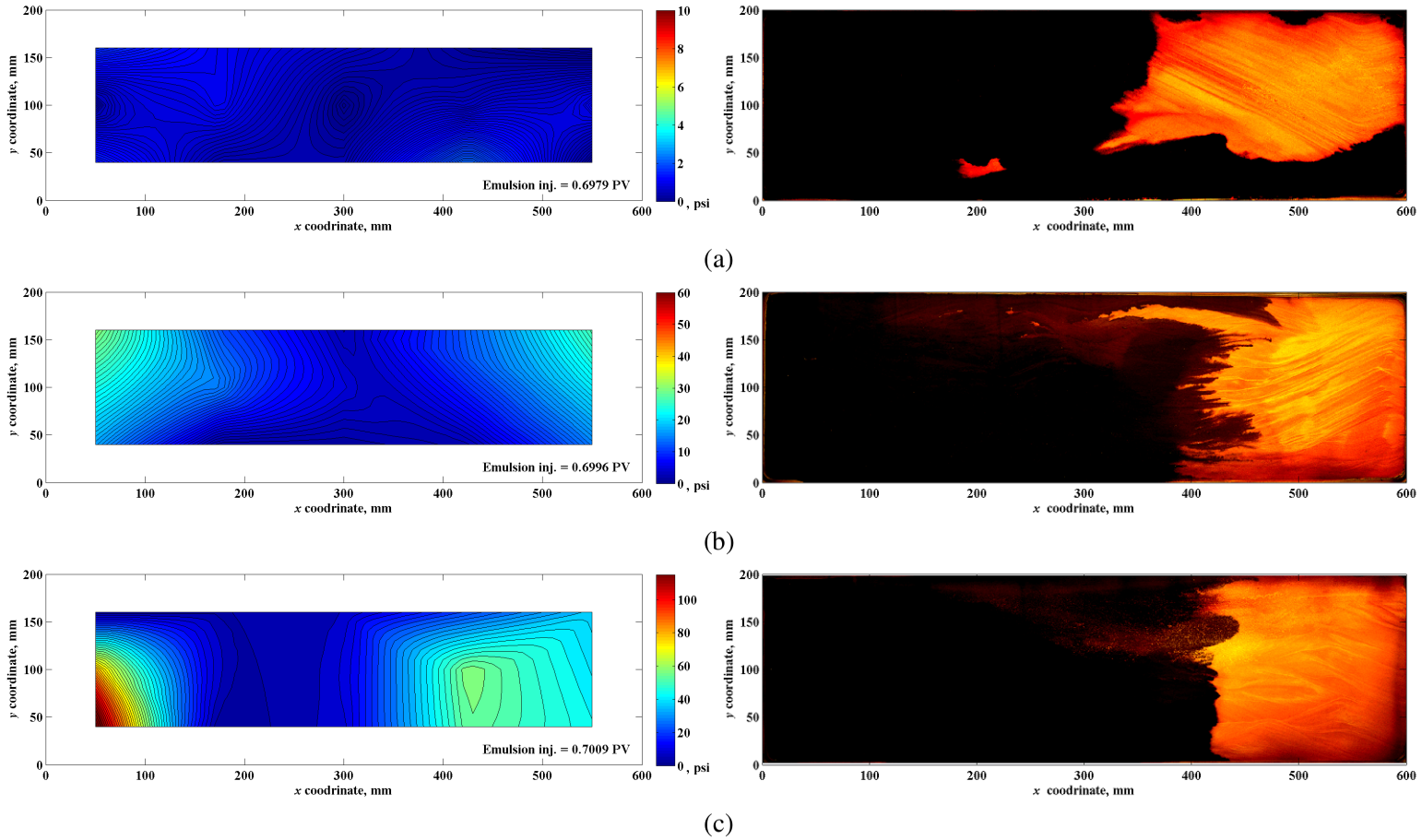


Figure 5.18: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

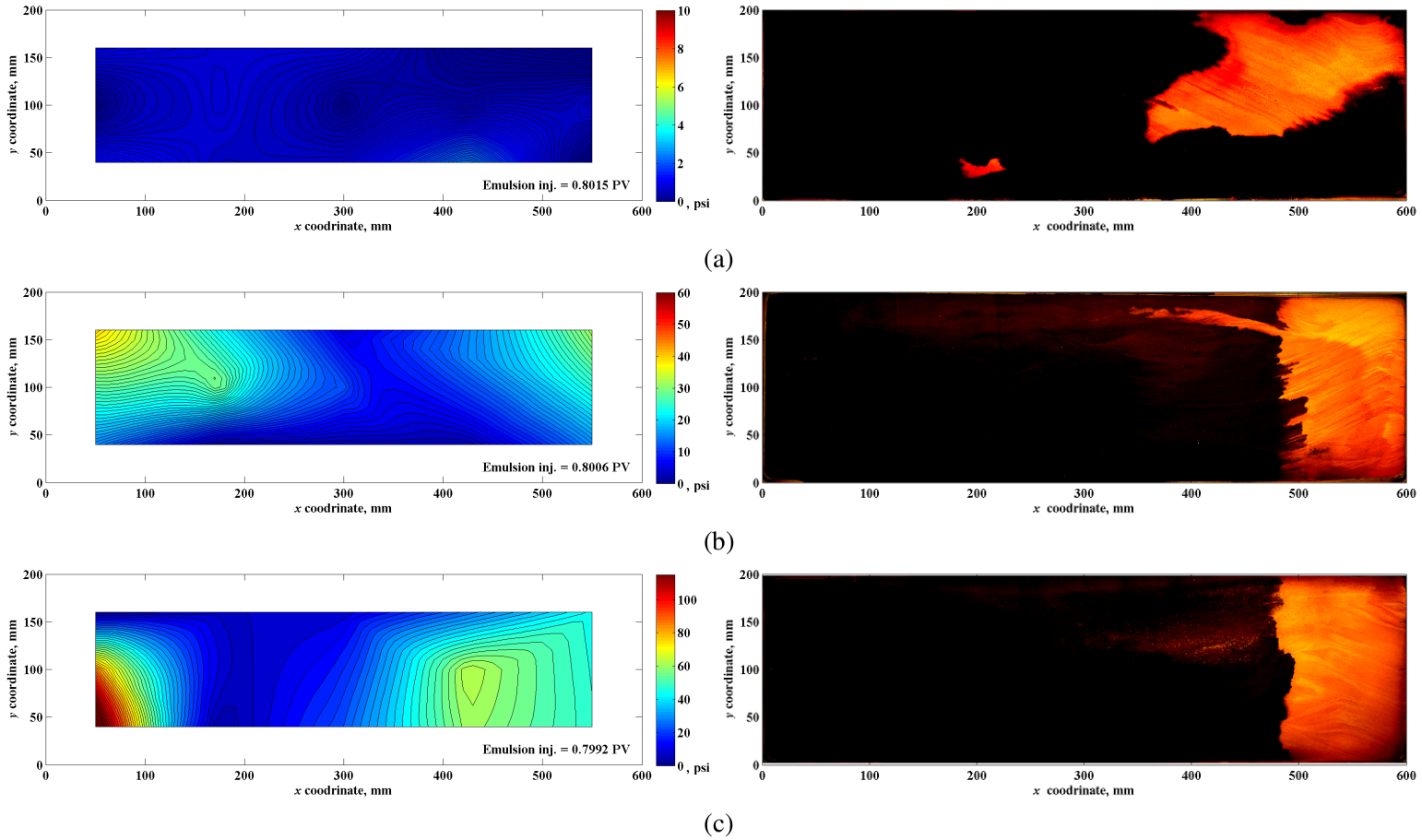


Figure 5.19: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

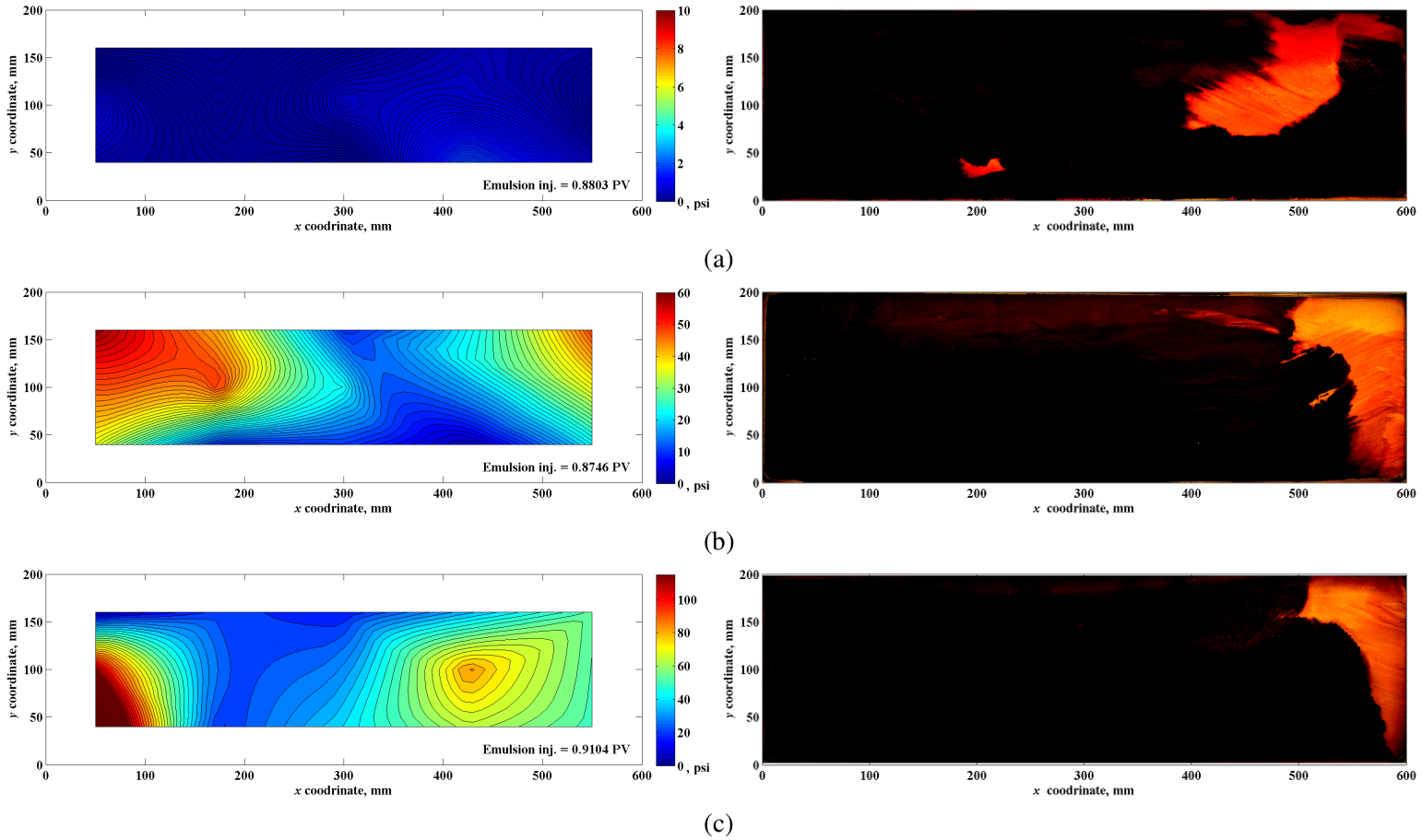


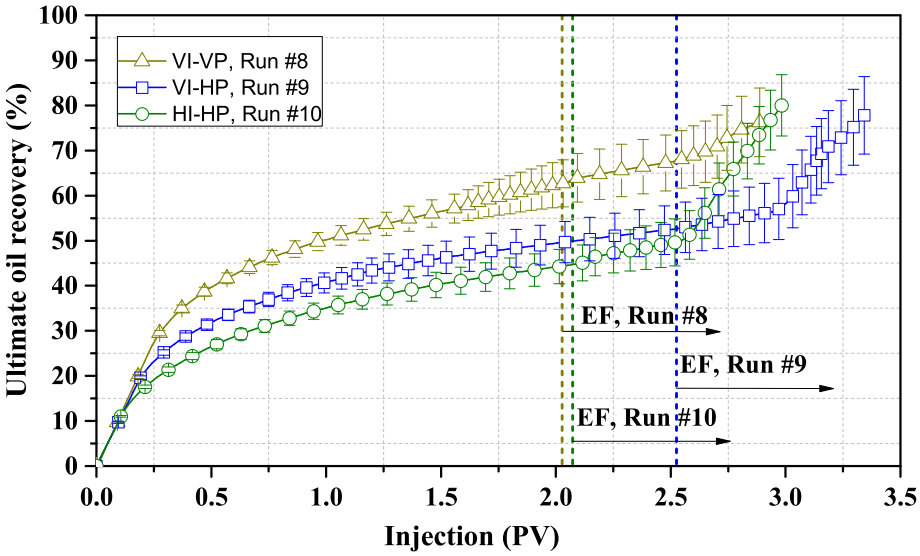
Figure 5.20: Comparison of pressure maps and cross-sectional images of sand packs in: (a) VI-VP; (b) VI-HP; (c) HI-HP configuration.

the flow direction. This can be due to the lack of data close to the boundaries where emulsion is possibly trying to slip through and builds up internal pressure. Due to the the lack of data points between pressure sensors and close to the boundaries, clear answer cannot be outlined.

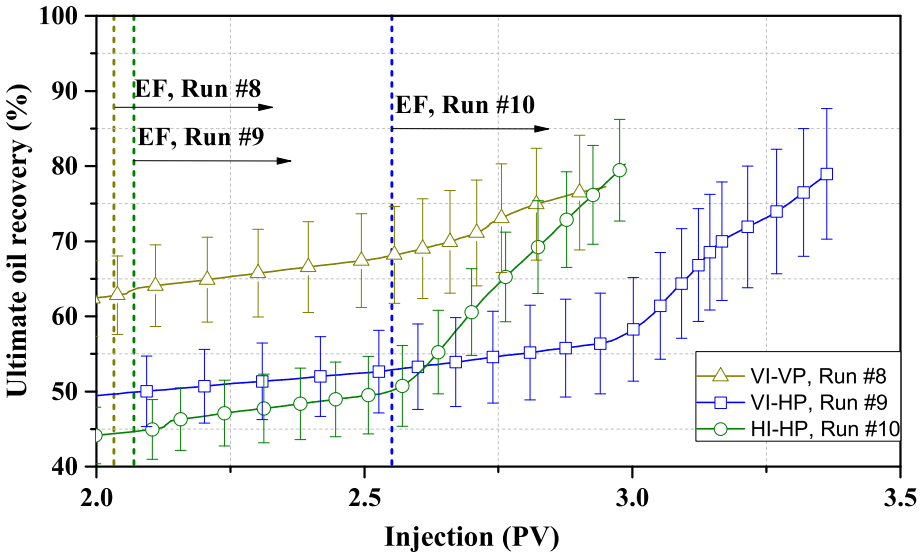
5.4.3 Effluent Analysis

Compared to the previous effluent analysis undertaken in previous experiments with 1D core holder, for the 2D core holder, the process of gravity segregation was stimulated using a centrifuge (Part #EW-17414-21, Cole-Parmer Canada, Inc.). The test tubes collected with fraction collector were centrifuged at 4000 rpm for one minute. The configuration with one minute and 4000 rpm was found experimentally as the most time-efficient.

Figure 5.21 (a) shows the ultimate oil recovery curves for the three different well configurations used in a 2D core holder. The data for recovery curves was collected from the test tubes after centrifuge. In order to compare data from different experiments, the injected volume was normalized using respected pore volume (PV). In terms of oil recovery during water flooding, it was found that VI-VP is the most efficient configuration. At least 19 % more can be recovered compare to others at 2 PV of water injection. This result mainly can be due to a delayed break through point and significantly larger distance between wells surfaces in horizontal direction.



(a)



(b)

Figure 5.21: Ultimate oil recovery curves for water flooding followed by emulsion flooding at different well configurations: (a) complete flooding process and (b) emulsion flooding process after water flooding.

Figure 5.21 (b) shows a zoomed view of the recovery trends. VI-VP configuration is not as efficient for the emulsion flooding as for the water flooding, only 14 % can be recovered at 0.9 PV emulsion injection. For the studied parameters, it is observed that the most efficient configuration for the emulsion flooding is HI-HP, almost 34.75 % of OOIP can be additionally recovered from reservoir. The ultimate oil recovery for VI-VP, VI-HP and HI-HP are 77.2 %, 78.9 % and 80.2 %, respectively. Emulsion injection was ceased for each run, once it appeared at the effluent side. There is a delay in oil recovery after the emulsion is introduced to the porous media. Oil and water require time to form new fronts inside the porous media and initially only water is present in the effluent.

Similar observations were made from image analysis and pressure mapping analysis. The lowest pressure drop across the core holder measured during emulsion flooding was found in VI-VP configuration, where emulsion flooding was found to be ineffective process in terms of oil recovery. While the pressure drop across the core holder gradually increases in VI-HP and HI-HP configurations up to 59.4 psi and 114.4 psi, respectively, and caused higher production rates. Hence, the *in-situ* pressure rise has connection to the efficiency of emulsion flooding process.

5.4.4 Quantitative Analysis

In order to estimate the behavior of emulsion flooding and provide a reasonable explanation for the velocity profiles, reference photographs are taken after oil saturation was completion and after water flooding completion at different configurations. These are shown in Figure 5.22. In addition, the last column shows evolution of the fractal interface of emulsion injection collected using commercial image analysis

software (Image-Pro Premier 9.1, Media Cybernetics, Inc.). The interface shows the boundaries of emulsion filled regions collected from each image. The background for the interface is post water flooding condition. The denser region means slower progress for the interface over singe time step.

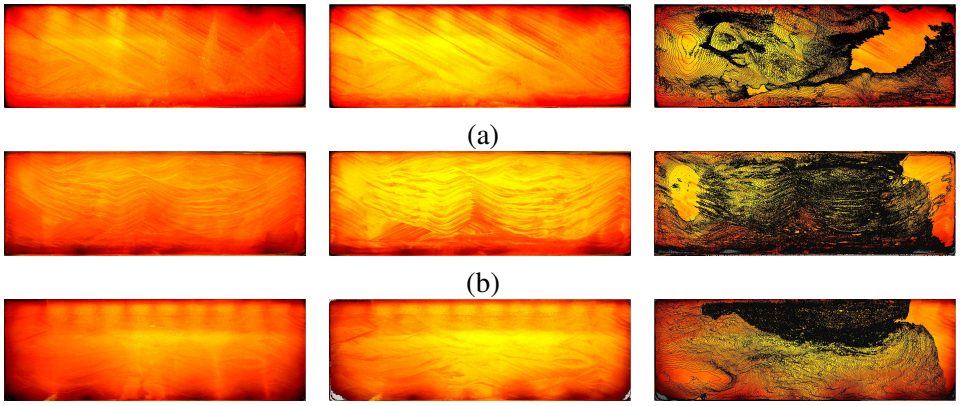


Figure 5.22: Left column shows sand pack prior to water flooding, central column shows sand pack prior emulsion flooding, right column shows evolution of interface of emulsion injection in (a) VI-VP configuration; (b) VI-HP configuration; (c) HI-HP configuration. Here, the direction of injection is from left to right.

After water flooding, most of the oil was washed out from central part of the core holder as represented by the yellow dyed water presented in Figure 5.22 (b). The least amount was for HI-HP, the most was for VI-VP, as it can be noticed on Figure 5.22 (a) and Figure 5.22 (c) and was previously confirmed with effluent analysis.

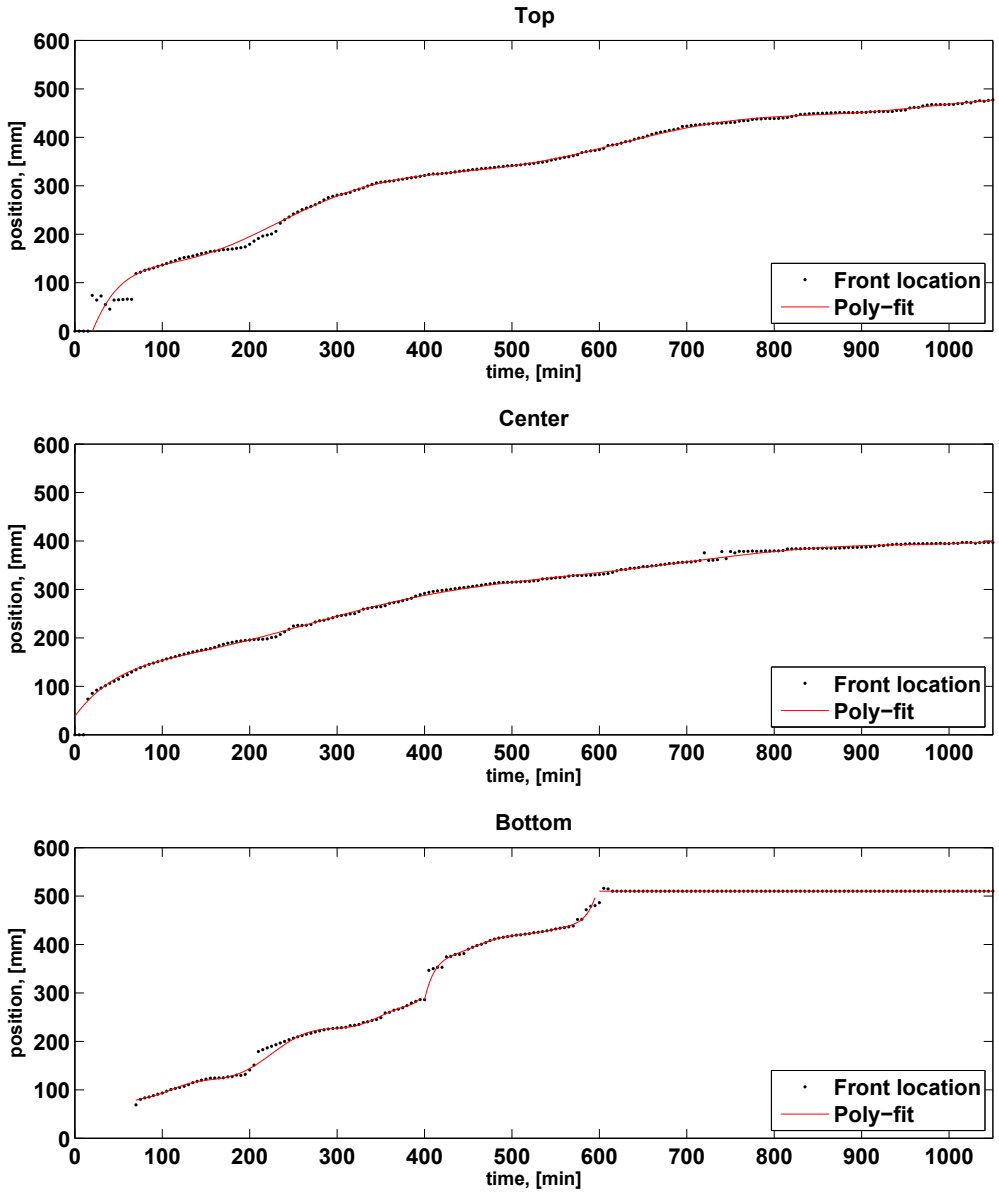


Figure 5.23: Emulsion front position versus time at three locations of a sand pack (a) top; (b) center; (c) bottom, as it is shown in reference Figure 5.10, for VI-VP configuration.

The position of the front plotted over the time for VI-VP configuration is shown in Figure 5.23. The position for the flood front follows a smooth curve for top and center locations. It also remains constant for bottom location at the 600 min due to a reflected flow from the right boundary. The order of cubic polynomial function fit is nine for top and eight for center location. For the bottom, the curve was divided into three regions, from 0 to 480 min, from 480 to 600 min and from 600 to ongoing. The orders for cubic polynomial function fits are nine, eight and seven, respectively.

The position of the front plotted over time for VI-HP configuration is shown in Figure 5.24. The position for the flood front follows smooth curve for top and center locations. There are some offset point at the top. The order for cubic polynomial function fit is nine for top and seven for center location. For the bottom, the curve was divided into two regions, from 0 to 480 min, from 480 to ongoing. The orders for cubic polynomial function fits are six and seven, respectively.

The position of the front plotted over time for HI-HP configuration is shown in Figure 5.25. The position for the flood front follows smooth curve for all three locations. The order for cubic polynomial function fit is 9, 7 and 6 for top, center and bottom location, respectively.

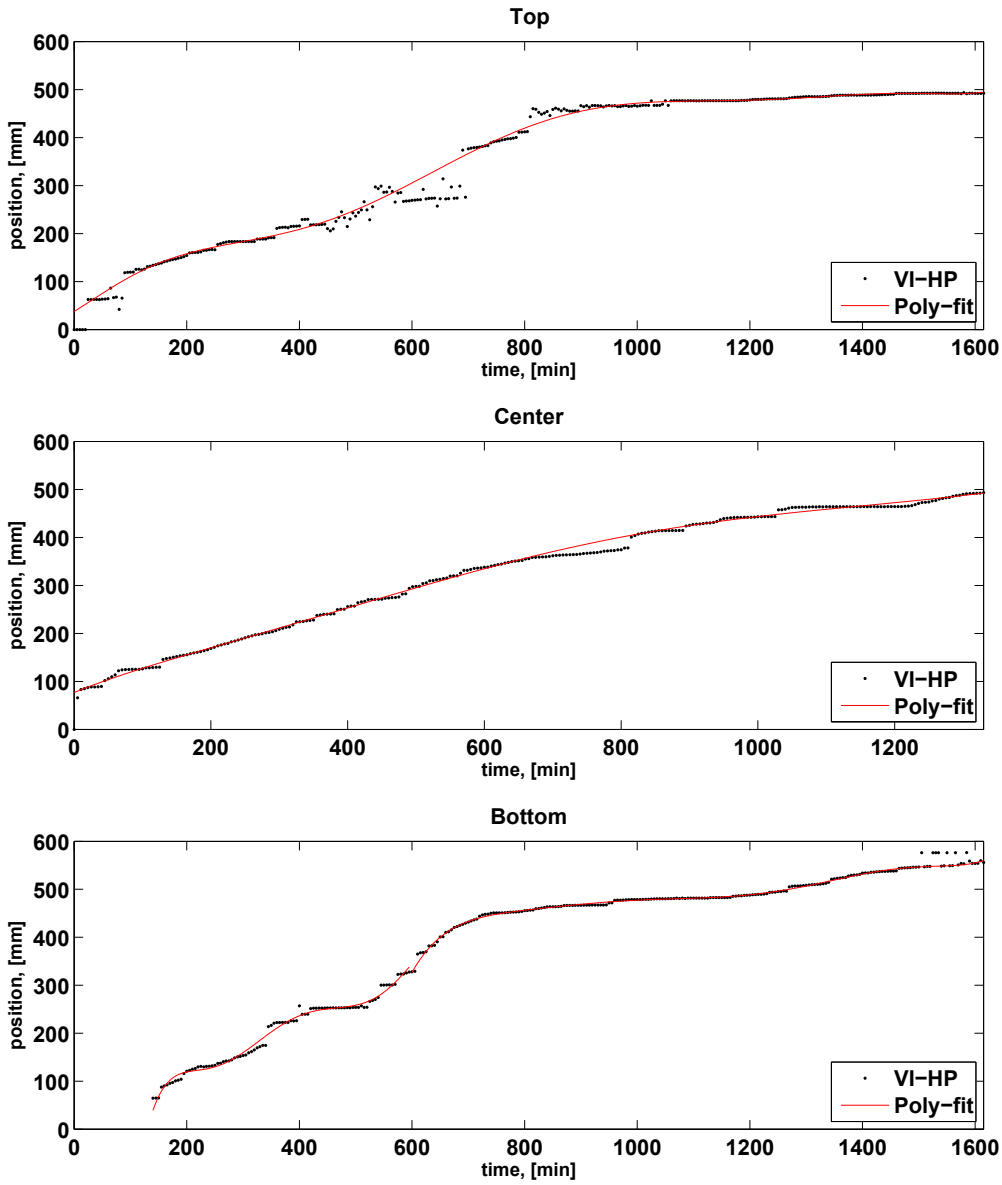


Figure 5.24: Emulsion front position versus time at three locations of a sand pack (a) top; (b) center; (c) bottom, as it is shown in reference Figure 5.10, for VI-HP configuration.

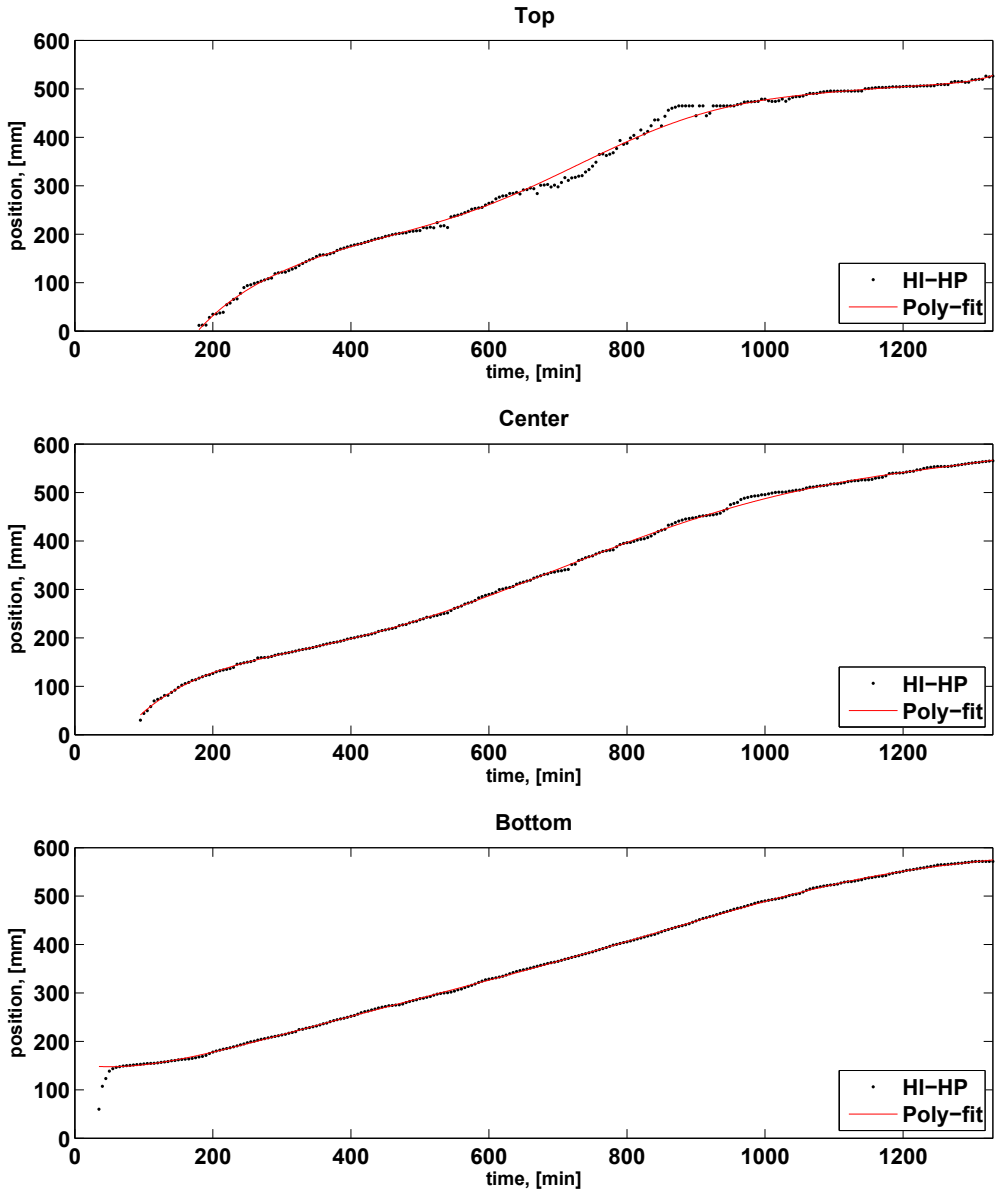


Figure 5.25: Emulsion front position versus time at three locations of a sand pack (a) top; (b) center; (c) bottom, as it is shown in reference Figure 5.10, for HI-HP configuration.

The derivatives of the collected cubic polynomial function fits were calculated to generate velocity profiles of the emulsion flood front. Figure 5.26 shows the velocity profiles versus volume of emulsion injected, collected at three locations, top, center and bottom, for each well configuration using custom developed code. The reference level for the velocity can be identified using parameters of sand packs from Table 5.2 and expected pore size distribution reported above. The superficial velocity, $U_{\text{ref,n}}$, taking into account emulsion injection flow rate, Q (1.0 cm^3/min), porosity, $\phi_{\text{run,n}}$, and geometry of sand pack (h - height and w - width) can be written as (Darby, 2001):

$$U_{\text{ref,n}} = \frac{Q}{h \cdot w \cdot \phi_{\text{run,n}}} \tag{5.1}$$

For Run #8, Run #9 and Run #10 it can be found that superficial velocity can be estimated as $1.177 \cdot 10^{-5}$m/s, $1.204 \cdot 10^{-5}$m/s, $1.215 \cdot 10^{-5}$m/s, respectively for the emulsion flooding process. For the water flooding process with flow rate equal to 1.75 cm/m^3 during Run #8, Run #9 and Run #10 the superficial velocity can be estimated as $2.06 \cdot 10^{-5}$m/s, $2.107 \cdot 10^{-5}$m/s, $2.126 \cdot 10^{-5}$m/s, respectively. These numbers are close to the values reported earlier in previous Chapter. The superficial velocities are plotted along with emulsion front velocity for all configurations as a reference velocities.

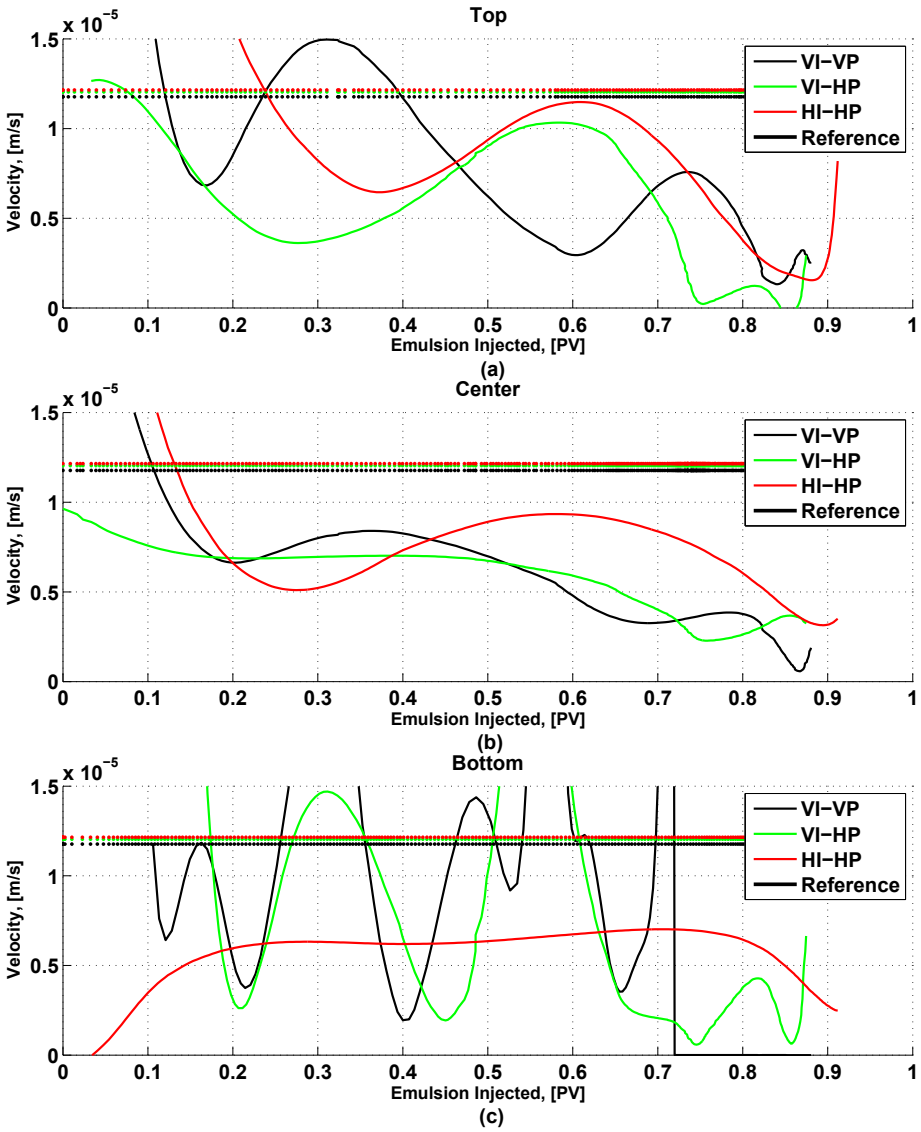


Figure 5.26: Emulsion flood front velocity at three locations of a sand pack (a) top; (b) center; (c) bottom, as it is shown in reference Figure 5.10

Velocity profiles, in Figure 5.26 (a), follow the same trend for all configurations. They fluctuated during the fronts movement across the core and slowly went down at the end of process as the front was getting closer to the production well. This can be due to presence of resident oil and its interactions with emulsion. It can be observed, that the profile was more stable for VI-HP configuration, as the least amount of oil can be observed in Figure 5.22.

Due to the presence of previously abandoned regions after water flooding at the center of the sand packs, as shown in Figure 5.22 central column, front velocity values are fluctuating close to the reference velocity level. The amplitude of fluctuations is smaller compared to the top profile, Figure 5.26 (b).

The most stable velocity profile was obtained for VI-HP configuration. Velocities for VI-VP and HI-HP follow similar trend with higher dispersion for HI-HP configuration. Compared to the top level, the velocities profiles behave more stable due to the presence of only a single liquid, water, in the porous media. By the end of emulsion flooding, front velocity stabilizes and all configurations end in the same region.

The most unstable trends for the velocity profiles were found at the bottom reference level for VI-VP and VI-HP configurations, where most of the resident oil was accumulated after water flooding, as shown in Figure 5.26 (c). However, HI-HP shows a smooth and stable curve with values close to the reference level. Horizontal injection for the emulsion allows the use of boundary conditions to guarantee improved stability for the front. From interface observation, Figure 5.22 the right column,it can be seen that emulsion progresses as piston like along the bottom surface without any instabilities. That observation can lead to the stable velocity

profile.

As it was found from the image analysis, pressure mapping and effluent analysis the most efficient and stable configuration for the water flooding followed by emulsion flooding is HI-HP. The stability of velocity profiles may also lead to the efficiency of enhanced oil recovery process, as it is also the most stable configuration.

5.5 Conclusion

The flow visualization and pressure mapping study has been carried out to investigate the effect from the well configuration on emulsion flooding after water flooding. Three different configurations for well orientation have been tested and analysed in terms of oil recovery and stability of the emulsion front. Visual analysis reports the pattern for the emulsion and shows *in-situ* processes happening inside porous media, such as oil trapping and channeling. Due to the presence of the red dye in oil, the region in front of emulsion turns into a red color during flow visualization. In a field test emulsion can be used to create regions with low permeability and the new well can be placed in front of the emulsion. It allows continue of oil recovery with traditional water flooding method.

Effluent analysis shows that vertical injection (VI) - vertical production (VP) is an efficient method for water flooding. However, emulsion flooding provides higher efficiency rate for the reservoirs with higher concentration for resident oil. It can be concluded that horizontal injection (HI) - horizontal production (HP) enhances oil recovery higher than any other configurations. The quantitative analysis suggests that stable velocity profile for an emulsion front can be achieved in HI-HP

configuration. While vertical injection (VI) - horizontal production (HP) and VI-VP has similar profiles to each other. It can be due to the same configuration for injection point. Obtained results can be used in future during field trials or in theory developments for the emulsion flooding.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This study was aimed to investigate the efficiency of using emulsion flooding as a recovery agent. After reviewing the literature for physical processes inside oil reservoirs, important parameters for reservoir characterization were identified. It was also found that emulsion produced *ex-situ* and *in-situ* can be used as flooding agent. However, due to similarities in emulsion properties to the resident oil and water, extensive post-experimental analysis is required. As a result, the first priority was identified to be developing a core flooding system, which can be used in future core flooding experiments. Extensive literature review outlined the existence of traditional core flooding system and their limitations. An improved experimental apparatus was designed and constructed using up-to-date available technologies in the market. In addition, the main component of the system, core holder, was also modified. Established design for the end plugs of core holder was improved using available computational resources. It was also shown that developed apparatus can be further upgraded if it is required to study other flooding regimes. The software with graphical user interface and remote alarming system was developed to

guarantee continuous control in experiments.

Having highly controlled experimental apparatus allowed to perform benchmark experiments with oil and water and to understand the limitations of developed systems. As a result of routine core flooding experiments, an automated method for effluent analysis was developed using real time readings from the downstream mass flow meter. This method allows to automate process of core flooding experiments and collect effluent data in efficient time manner. Using this method, first experiments for enhanced oil recovery were conducted with emulsion. It was found that *ex-situ* produced emulsion has high recovery rate and it is a feasible method to recover oil from porous structure. Up to 80 % of resident oil can be recovered using direct injection or after water flooding. However, taking into account costs for emulsion production, as it was pointed out earlier, new technique for the emulsion flooding must be developed.

An alternative flooding technique for the emulsion flooding was proposed, developed and tested. This technique bases on a cyclic flooding of emulsion and water slugs through porous media with specified ratio, similar to water and steam (CSS, WAG) or water and chemicals (CAG) flooding techniques. The process was termed as water-alternate-emulsion (WAE). Collected data showed that at least an additional 20 % of a resident oil can be recovered from a reservoir compare to traditional flooding processes. It also allows the reduction of the required amount of emulsion three times compared to the direct injection. For instance, the final ultimate oil recovery was found to be 82 % of OOIP for overall 0.3 PV of emulsion injection. Sand packs were examined after experiments to provide additional understanding for the *in-situ* processes. However, it was limited due to the fact that once

pack recovered from the core holder it cannot be placed back. Thus, it was concluded that future studies with utilizing equipment for nondestructive testing, such as two dimensional (2D) core holder, of the *in-situ* process are required for more detailed understanding the interactions between different liquids inside the porous media and how the recovery curve formed during WAE strategy.

Finally, experimental flow visualization and pressure mapping analysis of emulsion flooding with horizontal and vertical wells was carried out as next step of this research project. During preparation to those experiments, it was discovered that the given emulsion has a bi-dispersal structure. A literature review showed that such structure was not previously reported in the literature. This finding leads to the reviewing existing theory about an emulsion, since it effects on *in-situ* processes happening at micro-scale, size of the pore. Testing three different configurations provided initial understanding for the emulsion flooding process at the scale of 2D core holder, which is eight times bigger compared to biaxial core holder. The velocity profiles of front floods, pressure maps and recovery curves of ultimate oil recovery were extracted and used along with collected images of the side wall. It was pointed out that the most efficient configuration for the emulsion flooding is horizontal injection - horizontal production (HI-HP). Field test trials can be now undertaken for similar reservoir conditions and geological formations. However, there is still a broad range of questions which should be answer to get the confidence that emulsion is an economically valuable process for oil recovery.

6.2 Future Work

Further studies can be done as a continuation of the current study and listed below:

- The biaxial holder can be replace with a modified version developed for acoustic measurement technique [1]. Acoustic measurements potentially can provide current saturation levels for each liquid and result in collecting data required for relative permeability curves.

- To be closer to the reservoir conditions, the core block can be modified as it is shown in Chapter 3. Sand packs will have different pore structure and might result in efficiency of emulsion flooding. Collected data can be further applied in field applications.

- In order to reproduce the real world scenario, when the reservoir has non homogeneous strata with residual oil saturation distributed within it, various configurations for strata need to be studied as well. Taking pictures from one side of the 2D visualization cell provide a map for oil, water and emulsion saturations.

- As it was shown in Chapter 5, during emulsion flooding resident oil forms a new front. Having the possibility to install new well during an experiment can be used to study the hypothesis of resuming to traditional water flooding.

- Water-alternate-emulsion (WAE) flooding technique proved to be an efficient method for oil recovery. However, it is still unclear what kind of interactions

[1] Paper in progress. M. Cassiede, A. Baldygin, A. Stephen, J. M. Shaw, D. S. Nobes, and S. K. Mitra, "Acoustic measurement technique in application to the bio-conversion of coal", *Energy & Fuels*, 2014

present inside porous media. WAE flooding can be accomplished in 2D core holder to visualize *in-situ* process.

- Finally, the 3D core holder can be used to study complex porous media, with permeable and non-permeable zones, and complex well configurations, *e.g.* steering well.

Bibliography

F. F. Nazzal and M. R. Wiesner. Microfiltration of emulsions. *Water*, 68(7):1187–1191, 1996.

M. S. Dresselhaus and I. L. Thomas. Alternative energy technologies. *Nature*, 414 (6861):332–7, November 2001. ISSN 0028-0836. doi: 10.1038/35104599. URL http://www.ncbi.nlm.nih.gov/pubmed/11713539.

Omrpublic.iea.org. Oil market report, 2014. URL http://omrpublic.iea.org/.

E. Tzimas, A. Georgakaki, C. Garcia Cortes, and S.D. Peteves. Enhanced Oil Recovery using Carbon Dioxide in the European Energy System. Technical Report December, Institute for Energy, Petten, The Netherlands, 2005.

D. W. Green and G. P. Willhite. *Enhanced oil recovery*. Society of Petroleum Engineers, Richardson, TX USA, 1998.

Q. Liu, M. Dong, S. Ma, and Y. Tu. Surfactant enhanced alkaline flooding for Western Canadian heavy oil recovery. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 293(1-3):63–71, February 2007. ISSN 09277757. doi: 10.1016/j.colsurfa.2006.07.013. URL http://linkinghub.elsevier.com/retrieve/pii/S0927775706005231.

K. Asghari and P. Nakutnyy. Experimental Results of Polymer Flooding of Heavy Oil Reservoirs. *Proceedings of Canadian International Petroleum Conference*, pages 1–7, June 2008. doi: 10.2118/2008-189. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-2008-189&soc=PETSOC.

P. Shen, J. Wang, S. Yuan, T. Zhong, and X. Jia. Study of Enhanced-Oil-Recovery Mechanism of Alkali/Surfactant/Polymer Flooding in Porous Media From Experiments. *SPE Journal*, 14(2), June 2009. ISSN 1086-055X. doi: 10.2118/126128-PA. URL http://www.spe.org/ejournals/jsp/journalapp.jsp?pageType=Preview&jid=ESJ&mid=SPE-126128-PA&pdfChronicleId=09014762801b994b.

V. Alvarado and E. Manrique. Enhanced Oil Recovery: An Update Review. *Energies*, 3(9):1529–1575, August 2010. ISSN 1996-1073. doi: 10.3390/en3091529. URL http://www.mdpi.com/1996-1073/3/9/1529/.

A. Hart. The novel THAICAPRI technology and its comparison to other thermal methods for heavy oil recovery and upgrading. *Journal of Petroleum Exploration and Production Technology*, December 2013. ISSN 2190-0558. doi: 10.1007/s13202-013-0096-4. URL http://link.springer.com/10.1007/s13202-013-0096-4.

F. J. Hein, D. Leckie, S. Larter, and J. Suter. *Heavy-oil and Oil-sand Petroleum Systems in Alberta and Beyond: AAPG Studies in Geology 64*. Amer Assn of Petroleum Geologists, 1st edition, 2013. ISBN

0891810706. URL http://books.google.com/books?hl=en&lr=&id=VbkuAgAAQBAJ&pgis=1.

Www.altpetrol.com. Fuel oil emulsions, 2014. URL http://www.altpetrol.com/en/2d-pd-foe.html.

J. G. Speight. *The Chemistry and Technology of Petroleum*. CRC Press, 4 edition edition, 2006. ISBN 0849390672.

D. M. Adams. Experiences With Waterflooding Lloydminster Heavy-Oil Reservoirs. *Journal of Petroleum Technology*, 34(8), August 1982. ISSN 0149-2136. doi: 10.2118/10196-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00010196&soc=SPE.

S. R. Larter and I. M. Head. Oil Sands and Heavy Oil: Origin and Exploitation. *Elements*, 10(4):277–283, August 2014. ISSN 1811-5209. doi: 10.2113/gselements.10.4.277. URL http://elements.geoscienceworld.org/cgi/doi/10.2113/gselements.10.4.277.

M. Arhuoma, D. Yang, M. Dong, H. Li, and R. Idem. Numerical Simulation of Displacement Mechanisms for Enhancing Heavy Oil Recovery during Alkaline Flooding. *Energy & Fuels*, 23(12):5995–6002, December 2009. ISSN 0887-0624. doi: 10.1021/ef900690y. URL http://pubs.acs.org/doi/abs/10.1021/ef900690y.

F. Qiu and D. Mamora. Experimental Study of Solvent-Based Emulsion Injection to Enhance Heavy Oil Recovery in Alaska North Slope Area. In *Proceedings of Canadian Unconventional Resources and In-*

ternational Petroleum Conference, pages 1–16. Society of Petroleum Engineers, October 2010. ISBN 9781555633127. doi: 10.2118/136758-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-136758-MS&soc=SPE.

L. Schramm. *Emulsions, Foams and Suspensions: Fundamentals and Applications*. WILEY-VCH, Weinheim, 2006. ISBN 3-527-30743-5. doi: 10.1002/cphc.200500530. URL http://doi.wiley.com/10.1002/cphc.200500530.

D. T. Wasan, S. M. Shah, N. Aderangi, M. S. Chan, and J. J. McNamara. Observations on the Coalescence Behavior of Oil Droplets and Emulsion Stability in Enhanced Oil Recovery. *Society of Petroleum Engineers Journal*, 18(6), December 1978. ISSN 0197-7520. doi: 10.2118/6846-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00006846&soc=SPE.

C. McAuliffe. Crude-Oil-Water Emulsions to Improve Fluid Flow in an Oil Reservoir. *Journal of Petroleum Technology*, 25(6):721–726, June 1973. ISSN 0149-2136. doi: 10.2118/4370-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00004370&soc=SPE.

A. Mandal, A. Samanta, A. Bera, and K. Ojha. Role of oil-water emulsion in enhanced oil recovery. In *2010 International Conference on Chemistry and Chemical Engineering*, number Iccce, pages 190–194. IEEE, August 2010a. ISBN 978-1-4244-7765-4. doi: 10.1109/ICCCENG.2010.

5560393. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5560393.

A. Samanta, A. Bera, K. Ojha, and A. Mandal. Comparative studies on enhanced oil recovery by alkalisurfactant and polymer flooding. *Journal of Petroleum Exploration and Production Technology*, 2(2):67–74, June 2012. ISSN 2190-0558. doi: 10.1007/s13202-012-0021-2. URL http://link.springer.com/10.1007/s13202-012-0021-2.

M. Moradi, M. Kazempour, J. T. French, and V. Alvarado. Dynamic flow response of crude oil-in-water emulsion during flow through porous media. *Fuel*, 135:38–45, November 2014. ISSN 00162361. doi: 10.1016/j.fuel.2014.06.025. URL http://linkinghub.elsevier.com/retrieve/pii/S0016236114005912.

R. Kumar, E. Dao, and K. K. Mohanty. SPE 129914 Emulsion Flooding of Heavy Oil. In *SPE Journal*, number April, pages 24–28, 2010.

R. M. Decker and D. L. Flock. Thermal Stability And Application Of Emulsion Composed Blocking Agents For Steamflooding. *Journal of Canadian Petroleum Technology*, 27(4):69–78, 1988. ISSN 0021-9487. doi: 10.2118/88-04-05. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-88-04-05&soc=PETSOC.

S. Cobos, M. S. Carvalho, and V. Alvarado. Flow of oil - water emulsions through a constricted capillary. *International Journal of Multiphase Flow*, 35(6):507–515, June 2009. ISSN 03019322. doi: 10.1016/j.ijmultiphaseflow.2009.

02.018. URL http://linkinghub.elsevier.com/retrieve/pii/S0301932209000366.

N. J. Hadia, L. S. Chaudhari, A. Aggarwal, S. K. Mitra, M. Vinjamur, and R. Singh. Experimental and numerical investigation of one-dimensional waterflood in porous reservoir. *Experimental Thermal and Fluid Science*, 32(2): 355–361, November 2007a. ISSN 08941777. doi: 10.1016/j.expthermflusci.2007.04.009. URL http://linkinghub.elsevier.com/retrieve/pii/S0894177707000660.

V. Santosh, S. K. Mitra, M. Vinjamur, and R. Singh. Experimental and Numerical Investigations of Waterflood Profiles with Different Well Configurations. *Energy & Fuels*, 21(6):3353–3359, November 2007. ISSN 0887-0624. doi: 10.1021/ef0700376. URL http://pubs.acs.org/doi/abs/10.1021/ef0700376.

V. Santosh, S. K. Mitra, M. Vinjamur, and M. S. Kumar. Flow Visualization of Waterflooding with Horizontal and Vertical Wells. *Petroleum Science and Technology*, 26(15):1835–1851, September 2008. ISSN 1091-6466. doi: 10.1080/10916460701296418. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/10916460701296418&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

L. S. Chaudhari, N. J. Hadia, S. K. Mitra, and M. Vinjamur. Flow Visualization of Two-Phase Flow through Layered Porous Media. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, 33(10):948–958,

January 2011a. ISSN 1556-7036. doi: 10.1080/15567030903330702. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/15567030903330702&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

S. Bagci and F. Gumrah. An examination of steam-injection processes in horizontal and vertical wells for heavy-oil recovery. *Journal of Petroleum Science and Engineering*, 8(1):59–72, July 1992. ISSN 09204105. doi: 10.1016/0920-4105(92)90044-2. URL http://linkinghub.elsevier.com/retrieve/pii/0920410592900442.

N. J. Hadia, L. S. Chaudhari, S. K. Mitra, M. Vinjamur, and R. Singh. Experimental investigation of use of horizontal wells in waterflooding. *Journal of Petroleum Science and Engineering*, 56(4):303–310, April 2007b. ISSN 09204105. doi: 10.1016/j.petrol.2006.10.004. URL http://linkinghub.elsevier.com/retrieve/pii/S0920410506002270.

R. D'Elia-S and J. Ferrer-G. Emulsion Flooding of Viscous Oil Reservoirs. In *Proceedings of Fall Meeting of the Society of Petroleum Engineers of AIME*. Society of Petroleum Engineers, September 1973. ISBN 9781555637736. doi: 10.2118/4674-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00004674&soc=SPE.

P. Walstra. Principles of emulsion formation. *Chemical Engineering Science*, 48(2):333–349, January 1993. ISSN 00092509. doi: 10.1016/0009-2509(93)80021-H. URL http://linkinghub.elsevier.com/retrieve/pii/000925099380021H.

A. Mandal, A. Samanta, A. Bera, and K. Ojha. Characterization of Oil - Water Emulsion and Its Use in Enhanced Oil Recovery. *Industrial & Engineering Chemistry Research*, 49(24):12756–12761, December 2010b. ISSN 0888-5885. doi: 10.1021/ie101589x. URL http://pubs.acs.org/doi/abs/10.1021/ie101589x.

V. R. Guillen, M. S. Carvalho, and V. Alvarado. Pore Scale and Macroscopic Displacement Mechanisms in Emulsion Flooding. *Transport in Porous Media*, 94(1):197–206, April 2012. ISSN 0169-3913. doi: 10.1007/s11242-012-9997-9. URL http://www.springerlink.com/index/10.1007/s11242-012-9997-9.

R. Kumar, E. Dao, and K. Mohanty. Heavy-Oil Recovery by In-Situ Emulsion Formation. *SPE Journal*, (July 2011):24–28, 2012. ISSN 1086-055X. doi: 10.2118/129914-PA. URL http://www.spe.org/ejournals/jsp/journalapp.jsp?pageType=Preview&jid=ESJ&mid=SPE-129914-PA&pdfChronicleId=090147628025afe0.

J. Bryan and A. Kantzas. Potential for Alkali-Surfactant Flooding in Heavy Oil Reservoirs Through Oil-in-Water Emulsification. *Journal of Canadian Petroleum Technology*, 48(2), February 2009. ISSN 0021-9487. doi: 10.2118/09-02-37. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-09-02-37&soc=PETSOC.

P. H. Krumrine and F. J.S. Surfactant, Polymer, and Alkali Interactions in Chemical Flooding Processes. *Proceedings of SPE Oilfield and Geothermal Chemistry Symposium*, June 1983. doi: 10.2118/

11778-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00011778&soc=SPE.

F. Qiu. The Potential Applications in Heavy Oil EOR With the Nanoparticle and Surfactant Stabilized Solvent-Based Emulsion. In *Proceedings of Canadian Unconventional Resources and International Petroleum Conference*, pages 1–12. Society of Petroleum Engineers, October 2010. ISBN 9781555633127. doi: 10.2118/134613-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-134613-MS&soc=SPE.

G. L. Nogueira, M. S. Carvalho, and V. Alvarado. Dynamic Network Model of Mobility Control in Emulsion Flow Through Porous Media. *Transport in Porous Media*, March 2013. ISSN 0169-3913. doi: 10.1007/s11242-013-0151-0. URL http://link.springer.com/10.1007/s11242-013-0151-0.

E. Boye, P. Jr, A. Lohne, J. O. Helland, and G. Virnovsky. Relative permeabilities for two- and three- phase flow processes relevant to the depressurization of the Statfjord field. *The Society of Core Analysis*, pages 1–12, 2008.

A. Baldygin, D. S. Nobes, and S. K. Mitra. New Laboratory Core Flooding Experimental System. *Industrial & Engineering Chemistry Research*, 53(34): 13497–13505, August 2014a. ISSN 0888-5885. doi: 10.1021/ie501866e. URL http://pubs.acs.org/doi/abs/10.1021/ie501866e.

A. Baldygin, D. S. Nobes, and S. K. Mitra. Water-Alternate-Emulsion (WAE): A new technique for enhanced oil recovery. *Journal of Petroleum Science and Engineering*, 121:167–173, July 2014b. ISSN 09204105. doi: 10.1016/j.petrol.

2014.06.021. URL http://linkinghub.elsevier.com/retrieve/pii/S0920410514001776.

A. Baldygin, D. S. Nobes, and S. K. Mitra. Novel Technique for Core Flooding Experiments. In *The biennial CSME International Congress: Symposium on Advanced Energy Systems (CSME'12 - Symposium on AES)*, University of Manitoba, Canada, June 2012.

A. Baldygin, D. S. Nobes, and S. K. Mitra. IMECE2014-37248 Oil recovery from porous media using emulsion. In *Proceedings of the ASME 2014 International Mechanical Engineering Congress & Exposition*, page 6, Montreal, Quebec, Canada, 2014c. IMECE 2014.

H. Darcy. *Les fontaines publiques de la ville de Dijon*. Paris, 1856.

L F Athy. American association of petroleum geologists. 17(1):1–24, 1930.

B. F. Towler. *Fundamental principles of reservoir engineering. SPE TEXTBOOK SERIES Vol. 8*. 2002.

Edward W. Washburn. Note on a Method of Determining the Distribution of Pore Sizes in a Porous Material. *Proceedings of the National Academy of Sciences of the United States of America*, 7(4):115–116, 1921. URL http://www.jstor.org/stable/84084?seq=1#page_scan_tab_contents.

L. P. Dake. *Fundamentals of reservoir engineering*. Elsevier Science B.V., Amsterdam, seventeent edition, 1998.

S E Buckley and M C Leverett. Mechanism of Fluid Displacement in Sands. *Transactions of the AIME*, 146(1):107–116, 1941. doi: 10.2118/942107-G.

H.L. Stone. Probability Model for Estimating Three-Phase Relative Permeability. *Journal of Petroleum Technology*, 22(2), February 1970. ISSN 0149-2136. doi: 10.2118/2116-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00002116&soc=SPE.

H.L. Stone. Estimation of Three-Phase Relative Permeability And Residual Oil Data. *Journal of Canadian Petroleum Technology*, (May), 1973. ISSN 0021-9487. doi: 10.2118/73-04-06. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-73-04-06&soc=PETSOC.

A. T. Corey, C. H. Rathjens, J. H. Henderson, and M. R. J. Wyllie. Three-Phase Relative Permeability. *Journal of Petroleum Technology*, 8(11):63–65, November 1956. ISSN 0149-2136. doi: 10.2118/737-G. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-000737-G&soc=SPE.

M.R.J. Wyllie. Interrelationship between wetting and nonwetting phase relative permeability. *Transactions of the AIME*, page 83, 1961.

M C Leverett. Capillary behavior in porous solids. *Trans. Am. Inst*, 1940.

Edward W. Washburn. The Dynamics of Capillary Flow. *Physical Review*, 17(3): 273–283, March 1921. ISSN 0031-899X. doi: 10.1103/PhysRev.17.273. URL http://link.aps.org/doi/10.1103/PhysRev.17.273.

T.F. Moore and R.L. Slobod. Displacement of Oil by Water-Effect of Wettability, Rate, and Viscosity on Recovery. In *Fall Meeting of the Petroleum Branch of*

AIME. Society of Petroleum Engineers, April 2013. doi: 10.2118/502-G. URL http://www.onepetro.org/doi/10.2118/502-G.

R W S Foulser, S G Goodyear, and R J Sims. New concepts in relative permeabilities at high capillary numbers for surfactant flooding. *Transport in Porous Media*, 6(3):223–240, June 1991. ISSN 0169-3913. doi: 10.1007/BF00208951. URL http://link.springer.com/10.1007/BF00208951.

M. I. Romero, M. S. Carvalho, and V. Alvarado. Experiments and network model of flow of oil-water emulsion in porous media. *Physical Review E*, 84(4):1–7, October 2011. ISSN 1539-3755. doi: 10.1103/PhysRevE.84.046305. URL http://link.aps.org/doi/10.1103/PhysRevE.84.046305.

G M Homsy. Viscous fingering in porous media. *Annual Review of Fluid Mechanics*, 19(1):271–311, 1987.

W. R. Schowalter. Stability criteria for miscible displacement of fluids from a porous medium. *AIChE Journal*, 11(1):99–105, January 1965. ISSN 0001-1541. doi: 10.1002/aic.690110122. URL http://doi.wiley.com/10.1002/aic.690110122.

T. R. French, J. S. Broz, P. B. Lorenz, and K. M. Bertus. Use of Emulsions for Mobility Control During Steamflooding. In *SPE California Regional Meeting*. Society of Petroleum Engineers, April 1986. ISBN 978-1-55563-615-9. doi: 10.2118/15052-MS. URL https://www.onepetro.org/conference-paper/SPE-15052-MS?sort=&start=0&q=SPE+

15052&from_year=&peer_reviewed=&published_between=&fromSearchResults=true&to_year=&rows=10#.

G. W. Schmidt. Interstitial Water Composition and Geochemistry of Deep Gulf Coast Shales and Sandstones. *AAPG Bulletin*, 57(2):321–337, 1973. ISSN 0149-1423. URL http://archives.datapages.com/data/bulletns/1971-73/data/pg/0057/0002/0300/0321.htm.

S. Bagci, M. V. Kok, and U. Turksoy. SPE 65394 Effect of Brine Composition and Alkaline Fluid on the Permeability Damage of Limestone Reservoirs. 2001a.

S. Bagci, M. V. Kok, and U. Turksoy. Effect of brine composition on oil recovery by waterflooding. *Petroleum Science and Technology*, 19(3-4):359–372, January 2001b. ISSN 1091-6466. doi: 10.1081/LFT-100000769. URL http://www.informaworld.com/openurl?genre=article&doi=10.1081/LFT-100000769&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

H. O. Yildiz, M. Valat, and N. R. Morrow. Effect of Brine Composition On Wettability And Oil Recovery of a Prudhoe Bay Crude Oil. *Journal of Canadian Petroleum Technology*, 38(01), April 2013. ISSN 0021-9487. doi: 10.2118/99-01-02. URL https://www.onepetro.org/journal-paper/PETSOC-99-01-02.

M. Moradi, V. Alvarado, and S. Huzurbazar. Effect of Salinity on Water-in-Crude Oil Emulsion: Evaluation through Drop-Size Distribution Proxy. *Energy & Fu-*

els, 25(1):260–268, January 2011. ISSN 0887-0624. doi: 10.1021/ef101236h. URL http://pubs.acs.org/doi/abs/10.1021/ef101236h.

C. E. Perles, P. L. O. Volpe, and A. J. F. Bombard. Study of the Cation and Salinity Effect on Electrocoalescence of Water/Crude Oil Emulsions. *Energy & Fuels*, 26(11):121107122506002, November 2012. ISSN 0887-0624. doi: 10.1021/ef301433m. URL http://dx.doi.org/10.1021/ef301433m.

H. K. Sarma, B. B. Maini, and K. Jha. Evaluation of Emulsified Solvent Flooding For Heavy Oil Recovery. *Journal of Canadian Petroleum Technology*, 37(7):55–62, July 1998. ISSN 0021-9487. doi: 10.2118/98-07-06. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-98-07-06&soc=PETSOC.

H. A. Son, K. Y. Yoon, G. J. Lee, J. W. Cho, S. K. Choi, J. W. Kim, K. C. Im, H. T. Kim, K. S. Lee, and W. M. Sung. The potential applications in oil recovery with silica nanoparticle and polyvinyl alcohol stabilized emulsion. *Journal of Petroleum Science and Engineering*, November 2014. ISSN 09204105. doi: 10.1016/j.petrol.2014.11.001. URL http://www.sciencedirect.com/science/article/pii/S0920410514003593.

E. J. Windhab, M. Dressler, K. Feigl, P. Fischer, and D. Megias-Alguacil. Emulsion processingfrom single-drop deformation to design of complex processes and products. *Chemical Engineering Science*, 60(8-9):2101–2113, April 2005. ISSN 00092509. doi: 10.1016/j.ces.2004.12.003. URL http://linkinghub.elsevier.com/retrieve/pii/S000925090400884X.

B. Binks. *Modern Aspects of Emulsion Science*. Royal Society of Chemistry, Cambridge, 1998. ISBN 978-0-85404-439-9. doi: 10.1039/9781847551474. URL http://ebook.rsc.org/?DOI=10.1039/9781847551474.

G. P. Willhite. *Waterflooding*. Society of Petroleum Engineers, Richardson, TX USA, 1986. ISBN 1-55563-005-7.

L. A. Rapoport and W. J. Leas. SPE-000213-G Properties of linear waterfloods. *Petroleum Transactions AMIE*, 198:139–148, 1953.

N. J. Hadia, L. S. Chaudhari, S. K. Mitra, M. Vinjamur, and R. Singh. Waterflood Profiles and Oil Recovery with Vertical and Horizontal Wells. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, 30(17):1604–1618, January 2008a. ISSN 1556-7036. doi: 10.1080/15567030701773806. URL http://www.informaworld.com/openurl?genre=article&doi=10.1080/15567030701773806&magic=crossref||D404A21C5BB053405B1A640AFFD44AE3.

N. J. Hadia, S. K. Mitra, and M. Vinjamur. Estimation of permeability heterogeneity in limestone outcrop by pressure measurements: Experiments and numerical simulation. *Experimental Thermal and Fluid Science*, 40:177–184, March 2012. ISSN 08941777. doi: 10.1016/j.expthermflusci.2012.03.005. URL http://linkinghub.elsevier.com/retrieve/pii/S0894177712000726.

N. J. Hadia, L. S. Chaudhari, S. K. Mitra, M. Vinjamur, and R. Singh. Effect of Scaling Parameters on Waterflood Performance with Horizontal and Vertical

Wells. *Energy & Fuels*, 22(1):402–409, January 2008b. ISSN 0887-0624. doi: 10.1021/ef070097b. URL http://pubs.acs.org/doi/abs/10.1021/ef070097b.

R. D. Wyckoff and H. G. Botset. The Flow of Gas-Liquid Mixtures Through Unconsolidated Sands. *Physics*, 7(9):325, 1936. ISSN 01486349. doi: 10.1063/1.1745402. URL http://scitation.aip.org/content/aip/journal/jap/7/9/10.1063/1.1745402.

R. Nasralla, M. Alotaibi, and H. Nasr-El-Din. Efficiency of Oil Recovery by Low Salinity Water Flooding in Sandstone Reservoirs. In *Proceedings of SPE Western North American Region Meeting*, number 1967. Society of Petroleum Engineers, May 2011. ISBN 9781613991206. doi: 10.2118/144602-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-144602-MS&soc=SPE.

E. H. Legowo and S. W. Pratomo. Microbial Core Flooding Experiments Using Indigenous Microbes. In *Proceedings of SPE Asia Pacific Improved Oil Recovery Conference*, pages 1–11. Society of Petroleum Engineers, October 1999. doi: 10.2118/57306-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00057306&soc=SPE.

V. Hornof and N. R. Morrow. Gravity Effects in the Displacement of Oil by Surfactant Solutions. *SPE Reservoir Engineering*, 2(4), November 1987. ISSN 0885-9248. doi: 10.2118/13573-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00013573&soc=SPE.

S.-H. Chang and R. B. Grigg. Laboratory Flow Tests Used To Determine Reservoir Simulator Foam Parameters for EVGSAU CO2 Foam Pilot. In *Proceedings of Permian Basin Oil and Gas Recovery Conference*, pages 483–492. Society of Petroleum Engineers, March 1994. ISBN 9781555634735. doi: 10.2118/27675-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00027675&soc=SPE.

S. Sedaee and R. Fariborz. Experimental Investigation of Steam/Methane Flooding in a Heavy Oil Reservoir. In *Proceedings of SPE International Petroleum Conference in Mexico*, number 1987. Society of Petroleum Engineers, November 2004. ISBN 9781555639822. doi: 10.2118/91968-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00091968&soc=SPE.

M. Nobakht, S. Moghadam, and Y. Gu. Effects of Viscous and Capillary Forces on CO2 Enhanced Oil Recovery under Reservoir Conditions. *Energy & Fuels*, 21(6):3469–3476, November 2007. ISSN 0887-0624. doi: 10.1021/ef700388a. URL http://pubs.acs.org/doi/abs/10.1021/ef700388a.

S. K. Veerabhadrappa, A. Doda, J. J. Trivedi, and E. Kuru. On the Effect of Polymer Elasticity on Secondary and Tertiary Oil Recovery. *Industrial & Engineering Chemistry Research*, 52(51):18421–18428, December 2013a. ISSN 0888-5885. doi: 10.1021/ie4026456. URL http://pubs.acs.org/doi/abs/10.1021/ie4026456.

K. M. Ko, B. H. Chon, S. B. Jang, and H. Y. Jang. Surfactant flooding characteristics of dodecyl alkyl sulfate for enhanced oil recovery. *Journal of Industrial and*

Engineering Chemistry, 20(1):228–233, January 2014. ISSN 1226086X. doi: 10.1016/j.jiec.2013.03.043. URL http://linkinghub.elsevier.com/retrieve/pii/S1226086X13001494.

C. J. Stanley. Perforated end plug plate for testing core samples, 1984.

C. J. Stanley. Porous end plug disk for testing core samples, 1985.

Www.corelab.com. Standard Core Holder - HCH Series, 2014a. URL http://www.corelab.com/cli/core-holders/standard-core-holder-hch-series.

Www.corelab.com. Core Laboratories: Hassler Type Core Holder - RCH Series, 2014b. URL http://www.corelab.com/cli/core-holders/hassler-type-core-holders-rch-series.

Www.coretest.com. CHG Series Coreholders, 2014. URL http://www.coretest.com/product_detail.php?p_id=58.

W. Chen, J. Peng, Y. Su, L. Zheng, L. Wang, and Z. Jiang. Separation of oil/water emulsion using Pluronic F127 modified polyethersulfone ultrafiltration membranes. *Separation and Purification Technology*, 66(3): 591–597, May 2009. ISSN 13835866. doi: 10.1016/j.seppur.2009.01.009. URL http://linkinghub.elsevier.com/retrieve/pii/S1383586609000148.

L. M. Fidalgo, G. Whyte, D. Bratton, C. F. Kaminski, C. Abell, and W. T. S. Huck. From microdroplets to microfluidics: selective emulsion separation in microfluidic devices. *Angewandte Chemie (International ed. in English)*, 47(11):

2042–5, January 2008. ISSN 1521-3773. doi: 10.1002/anie.200704903. URL http://www.ncbi.nlm.nih.gov/pubmed/18264960.

Patrick Brunelle (Quadrise Canada Corp.). Data sheet: UofA E2 Concentrate. *Personal communication*, 2012.

Manus Abrasive Systems. Bulk density chart, 2014. URL http://www.manusabrasive.com/bulk-density-chart/.

L. J. Klinkenberg. The permeability of porous media to liquids and gases. *American Petroleum Institute*, (02):57–73, 1941. ISSN 22186867. doi: 10.5510/OGP20120200114. URL http://www.socar.az/ogpi/files/uploader/57-73_Klinker.pdf.

N. S. K. Gunda, B. Bera, N. K. Karadimitriou, S. K. Mitra, and S. M. Hassanizadeh. Reservoir-on-a-chip (ROC): a new paradigm in reservoir engineering. *Lab on a chip*, 11(22):3785–92, November 2011. ISSN 1473-0189. doi: 10.1039/c1lc20556k. URL http://www.ncbi.nlm.nih.gov/pubmed/22011687.

S. Doorwar and K. Mohanty. Viscous Fingering during Non-Thermal Heavy Oil Recovery. In *Proceedings of SPE Annual Technical Conference and Exhibition*, number November. Society of Petroleum Engineers, October 2011. ISBN 9781613991473. doi: 10.2118/146841-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-146841-MS&soc=SPE.

A. Shah, R. Fishwick, J. Wood, G. Leeke, S. Rigby, and M. Greaves. A re-

view of novel techniques for heavy oil and bitumen extraction and upgrading. *Energy & Environmental Science*, 3(6):700, 2010. ISSN 1754-5692. doi: 10.1039/b918960b. URL http://xlink.rsc.org/?DOI=b918960b.

P. Luo, Y. Zhang, and S. Huang. A promising chemical-augmented WAG process for enhanced heavy oil recovery. *Fuel*, 104:333–341, February 2013. ISSN 00162361. doi: 10.1016/j.fuel.2012.09.070. URL http://linkinghub.elsevier.com/retrieve/pii/S0016236112007831.

E. T. S. Huang and L. W. Holm. Effect of WAG Injection and Rock Wettability on Oil Recovery During CO2 Flooding. *SPE Reservoir Engineering*, 3(1):119–129, February 1988. ISSN 0885-9248. doi: 10.2118/15491-PA. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=00015491&soc=SPE.

G. R. Scott. Comparison of CSS and SAGD Performance in the Clearwater Formation at Cold Lake. In *SPE International Thermal Operations and Heavy Oil Symposium and International Horizontal Well Technology Conference*. Society of Petroleum Engineers, April 2013. doi: 10.2118/79020-MS. URL http://www.onepetro.org/doi/10.2118/79020-MS.

S. Bagci and E. Tuzunoglu. 3D Model Studies of the Immiscible CO2 Process Using Horizontal Wells For Heavy Oil Recovery. In *Annual Technical Meeting*, page 12. Petroleum Society of Canada, April 1998. ISBN 9781613991022. doi: 10.2118/98-74. URL https://www.onepetro.org/conference-paper/PETSOC-98-74http://www.onepetro.org/doi/10.2118/98-74.

E. Tüzünoglu and S. Bagci. Scaled 3-D model studies of immiscible CO2 flooding using horizontal wells. *Journal of Petroleum Science and Engineering*, 26(1-4):67–81, May 2000. ISSN 09204105. doi: 10.1016/S0920-4105(00)00022-X. URL http://linkinghub.elsevier.com/retrieve/pii/S092041050000022X.

J. R. Christensen, E. H. Stenby, and A. Skauge. Review of WAG Field Experience. *SPE Reservoir Evaluation & Engineering*, 4(02):97–106, April 2013. ISSN 1094-6470. doi: 10.2118/71203-PA. URL http://www.onepetro.org/doi/10.2118/71203-PA.

W. S. Tortike. A Numerical Study Of Pressure Cycling Athabasca Oil Sand With Steam And Carbon Dioxide. In *Proceedings of Annual Technical Meeting*, pages 1–14. Society of Petroleum Engineers, April 1991. ISBN 9781555634827. doi: 10.2118/91-11. URL https://www.onepetro.org/conference-paper/PETSOC-91-11.

M. R. Islam and S. M. Farouq Ali. New Scaling Criteria For Polymer, Emulsion And Foam Flooding Experiments. *Journal of Canadian Petroleum Technology*, 28(4):79–87, 1989. ISSN 0021-9487. doi: 10.2118/89-04-05. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-89-04-05&soc=PETSOC.

M. R. Islam and S. M. Farouq Ali. New scaling criteria for in-situ combustion experiments. *Journal of Petroleum Science and Engineering*, 6(4):367–379, January 1992. ISSN 09204105. doi: 10.1016/0920-4105(92)

90063-7. URL http://linkinghub.elsevier.com/retrieve/pii/0920410592900637.

J. H. Abou-Kassem and S. M. Farouq Ali. Modelling of Emulsion Flow In Porous Media. *Journal of Canadian Petroleum Technology*, 34(6):30–38, June 1995. ISSN 0021-9487. doi: 10.2118/95-06-02. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=PETSOC-95-06-02&soc=PETSOC.

S. Akin, S. Bagci, and M. V. Kok. Dry Forward Combustion with Diverse Well Configurations. *Proceedings of SPE/AAPG Western Regional Meeting*, June 2000. doi: 10.2523/62551-MS. URL http://www.spe.org/elibrary/servlet/spepreview?id=00062551.

D. Tiab and E. C. Donaldson. *Petrophysics*. Gulf Professional Publishing, 3 edition, 2004. ISBN 9780750677110. doi: 10.1016/B978-075067711-0/50005-0. URL http://www.sciencedirect.com/science/article/pii/B9780750677110500050.

P. R. Waghmare and S. K. Mitra. Contact angle hysteresis of microbead suspensions. *Langmuir : the ACS journal of surfaces and colloids*, 26(22):17082–9, November 2010. ISSN 1520-5827. doi: 10.1021/la1025526. URL http://www.ncbi.nlm.nih.gov/pubmed/20886898.

K. S. Sorbie and Y. Huang. Rheological and transport effects in the flow of low-concentration xanthan solution through porous media. *Journal of Colloid and Interface Science*, 145(1):74–89, August 1991. ISSN 00219797. doi: 10.1016/

0021-9797(91)90100-M. URL http://linkinghub.elsevier.com/retrieve/pii/002197979190100M.

A. M. S. Maia, R. Borsali, and R. C. Balaban. Comparison between a polyacrylamide and a hydrophobically modified polyacrylamide flood in a sandstone core. *Materials Science and Engineering: C*, 29(2):505–509, March 2009. ISSN 09284931. doi: 10.1016/j.msec.2008.09.018. URL http://linkinghub.elsevier.com/retrieve/pii/S0928493108002324.

T. N. Castro Dantas, V. C. Santanna, A. A. Dantas Neto, F. D. S. Curbelo, and A. I. C. Garnica. Methodology to break test for surfactant-based fracturing gel. *Journal of Petroleum Science and Engineering*, 50(3-4): 293–298, March 2006. ISSN 09204105. doi: 10.1016/j.petrol.2005.12.001. URL http://linkinghub.elsevier.com/retrieve/pii/S0920410505001865.

R. Darby. *Chemical Engineering Fluid Mechanics*. Marcel Dekker, Inc., second edi edition, 2001.

S. M. Mahmood and W. E. Brigham. Two dimensional displacement of oil by gas and surfactant solution under foaming conditions. Technical report, United States Department of Energy, 1987.

R. V. Ponce F., M. S. Carvalho, and V. Alvarado. Oil recovery modeling of macroemulsion flooding at low capillary number. *Journal of Petroleum Science and Engineering*, 119:112–122, July 2014. ISSN 09204105. doi: 10.1016/j.petrol.

2014.04.020. URL http://linkinghub.elsevier.com/retrieve/pii/S0920410514001089.

G. P. Willhite, D. W. Green, D. M. Okoye, and M. D. Looney. A Study of Oil Displacement by Microemulsion Systems Mechanisms and Phase Behavior. *Society of Petroleum Engineers Journal*, 20(6), December 1980. ISSN 0197-7520. doi: 10.2118/7580-PA. URL https://www.onepetro.org/journal-paper/SPE-7580-PA.

C. Romero, B. Bazin, A. Zaitoun, and F. Leal Calderon. Behavior of a Scale Inhibitor Water-in-Oil Emulsion in Porous Media. In *Proceedings of International Symposium and Exhibition on Formation Damage Control*. Society of Petroleum Engineers, February 2006. ISBN 9781555632397. doi: 10.2118/98275-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-98275-MS&soc=SPE.

W. Anderson, D. Kozak, V. a Coleman, Å. K Jämting, and M. Trau. A comparative study of submicron particle sizing platforms: accuracy, precision and resolution analysis of polydisperse particle size distributions. *Journal of colloid and interface science*, 405:322–30, September 2013. ISSN 1095-7103. doi: 10.1016/j.jcis.2013.02.030. URL http://www.ncbi.nlm.nih.gov/pubmed/23759321.

S. Mei, J. Bryan, and A. Kantzas. Experimental Study of the Mechanisms in Heavy Oil Waterflooding Using Etched Glass Micromodel. *Proceedings of SPE Heavy Oil Conference Canada*, June 2012. doi: 10.2118/

157998-MS. URL http://www.onepetro.org/mslib/servlet/onepetropreview?id=SPE-157998-MS&soc=SPE.

T. Robert, R. Martel, S. H. Conrad, R. Lefebvre, and U. Gabriel. Visualization of TCE recovery mechanisms using surfactant-polymer solutions in a two-dimensional heterogeneous sand model. *Journal of contaminant hydrology*, 86 (1-2):3–31, June 2006. ISSN 0169-7722. doi: 10.1016/j.jconhyd.2006.02.013. URL http://www.ncbi.nlm.nih.gov/pubmed/16624443.

L. S. Chaudhari, N. J. Hadia, S. K. Mitra, and M. Vinjamur. Flow Visualization of Two-Phase Flow through Layered Porous Media. *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, 33(10):948–958, March 2011b. ISSN 1556-7036. doi: 10.1080/15567030903330702. URL http://www.tandfonline.com/doi/abs/10.1080/15567030903330702.

K. Naderi and T. Babadagli. Visual analysis of immiscible displacement processes in porous media under ultrasound effect. *Physical Review E*, 83(5):056323, May 2011. ISSN 1539-3755. doi: 10.1103/PhysRevE.83.056323. URL http://link.aps.org/doi/10.1103/PhysRevE.83.056323.

R. M. Butler and K. H. Chung. Goemetrical effect of steam injection on the formation of emulsions in the steam-assisted gravity drainage process. *The journal of Canadian Petroleum*, 1988.

W. Li and D. D. Mamora. SPE 133277 Experimental Investigation of Solvent Co-Injection in Vapor and Liquid Phase to Enhance SAGD Performance. *North*, i: 1–14, 2010.

B. Zehm. Software :: SendEmail - Send email with this free command line email client, 2009. URL http://caspian.dotconf.net/menu/Software/SendEmail/.

S. K. Veerabhadrappa, J. J. Trivedi, and E. Kuru. Visual Confirmation of the Elasticity Dependence of Unstable Secondary Polymer Floods. *Industrial & Engineering Chemistry Research*, 52(18):6234–6241, May 2013b. ISSN 0888-5885. doi: 10.1021/ie303241b. URL http://pubs.acs.org/doi/abs/10.1021/ie303241b.

G. E. King. Perforating Basics. How the perforating processes work. Technical report, 2009.

C. Cosad. Choosing a Perforation Strategy. *Oilfield Review*, 4(4):54–69, 1992.

R. A. Parrott and I. C. Walton. Perforating technology: well perforating solutions redefine sand management strategies. *Trade Publication*, 61(7):74, 2001.

B. A. Voll, B. M. Richard, A. D. Gabrysch, R. Kingwood, and A. Dale. Wire mesh filter, 1998. URL http://www.google.com/patents/US5849188.

Edge Detection. URL http://www.mathworks.com/help/vision/ref/edgedetection.html.

Patrick Brunelle (Quadrise Canada Corp.). UofA E2 Concentrate. *Personal communication*, page 5, 2012.

R J Blackwell, J R Rayne, and W M Terry. Factors Influencing the Efficiency of Miscible Displacement. *Society of Petroleum Engineers*, 1959.

D. Weitz, J. Stokes, R. Ball, and A. Kushnick. Dynamic Capillary Pressure in Porous Media: Origin of the Viscous-Fingering Length Scale. *Physical Review Letters*, 59(26):2967–2970, December 1987. ISSN 0031-9007. doi: 10.1103/PhysRevLett.59.2967. URL http://link.aps.org/doi/10.1103/PhysRevLett.59.2967.

Appendix

A-1 1D and 2D core holder

To ensure safe operation for the core holder when the system is pressurized, the components passed strength analysis for safe operation.

A-1.1 One dimensional core holder

Thread

The strength analysis on 2.500 x 4 UNC thread was performed to determine the safety factor at working conditions. Based on available information, the following conservative assumptions were made:

- The material's mechanical properties are homogeneous and isotropic throughout the cross-section;
- The machanical properties of the material are of equal magnitude in tension and compression;
- Cross-sections remain plane under loading conditions;
- The materials have ductile characteristics;
- Simple axial/shear loading results in uniform axial/shear stress distribution across a uniform crosssection;
- Materials obey Hooke's Laws of elasticity;
- The minimum yield strength at 0.2 % offset strain, rather than typical (average) yield strength of a material, is the basis by which stress at yield is defined;

- Shear strength is 50 % to 60 % of the minimum yield strength. Two existing theories are the distortion energy theory (0.577 * Ys) and the maximum shear stress theory (0.5 * Ys);
- The selected material will not be used in an environment that will adversely affect its mechanical or physical properties;
- Calculated stresses are based on static loading conditions if appropriate to the application;
- Material for body is 630 SS, E = 196 GPa.

The tensile strength area of screw thread can be calculated using the following formula:

$$A_t = 0.7854 \cdot \left(D - \frac{0.974}{n} \right)^2 \cdot \text{in}^2 = 4.0 \text{ , in}^2 \qquad \text{(A-1)}$$

where *D* - basic major diameter, *n* - number of thread per inch, and A_t - tensile strength area of screw thread.

Pressure on a thread is a result of pressure applied on end plug (1,500 psi) with outside diameter equal to 1.5 in. Resulted force on a thread is 2.651 x 10^3 lbf, and shear stress on thread can be found as:

$$\tau_{cut} = \frac{Q}{A_t} = \frac{2.651 \cdot 10^4 \text{ lbf}}{4.0 \text{ in}^2} = 662.9 \text{ , psi} \qquad \text{(A-2)}$$

so that safety factor is:

$$sf = \frac{0.577 \cdot Ys}{\tau_{cut}} = \frac{9.791 \cdot 10^4 \text{ psi}}{662.9 \text{ psi}} = 147.7 \qquad \text{(A-3)}$$

Obtained number for safety factor can guarantee safe operation at 1,500 psi.

Wall thickness

The strength analysis on a wall thickness was performed to determine the safety factor at working conditions. In order to estimate safety factor for body's wall, the ASME B31.3 standard has been used. The input parameters are:

- Outside diameter (D), 3.346 in.
- Inside diameter (ID), 2.550 in.
- Wall thickness (t), 0.398 in.
- Wall thickness in weakest point (next to thread) (t_{low}), 0.280 in.
- Stress value for material for Grad 316; 304; 321 (S), 20,000 psi
- Quality factor (E), 1
- Coefficient (function of material and temperature) (Y), 0.4
- Internal gauge pressure (P)

Formula from maximum allowed pressure from ASME B31.3 standard is:

$$P = \frac{2tSE}{D - 2tY} = 5258.3 \text{ , psi} \qquad \text{(A-4)}$$

or for the thinnest point:

$$P_{low} = \frac{2t_{low}SE}{D - 2t_{low}Y} = 3587.4 \text{ , psi} \qquad \text{(A-5)}$$

Taking into account working pressure, P_w, as 1,500 psi, safety factor is:

$$sf = \frac{P}{P_w} = 3.5 \qquad \text{(A-6)}$$

or for weakest point:

$$sf = \frac{P_low}{P_w} = 2.4 \qquad \text{(A-7)}$$

Both safety factors are above 2; thus, current dimensions for the core holder can safely be used up to 1,500 psi.

A-1.2 Two dimensional core holder

Wall thickness and deformation

The required thickness for the cast acrylic wall was estimated using commercial software to guarantee least deformation at the center of the sand pack at 300, 400 and 500 psi. The following boundary conditions have been chosen for model in SolidWorks Simulations:

- No penetration between parts;
- Holes for brackets were fixed;
- Pressure was applied on internal surfaces;
- "Counterbore with Nut" condition with head diameter (0.625 in), nominal shank diameter (0.4375 in) and axial load 100 lbf. for nut and bolt connections at front panel
- "Counterbore Screw" condition with head diameter (0.5 in), nominal shank diameter (0.3125 in) and axial load 80 lbf. for screws used to tight top cap.

For 1.5 in. wall thickness the deformations are 0.109, 0.145 and 0.181 in, respectively, as it is shown in Figures A-1 - A-3.

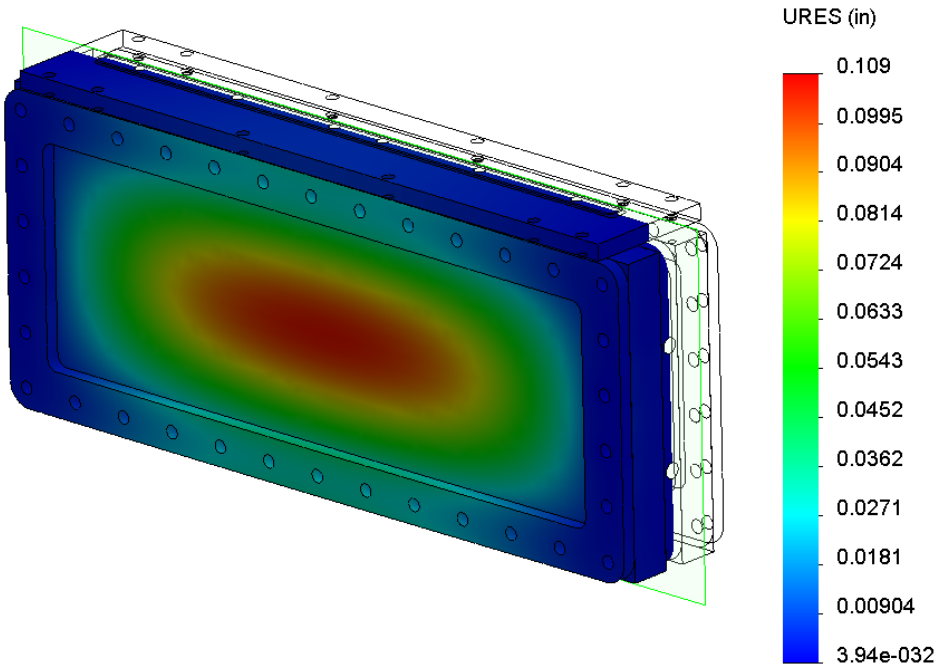


Figure A-1: Cast acrylic wall deformation at 300 psi.

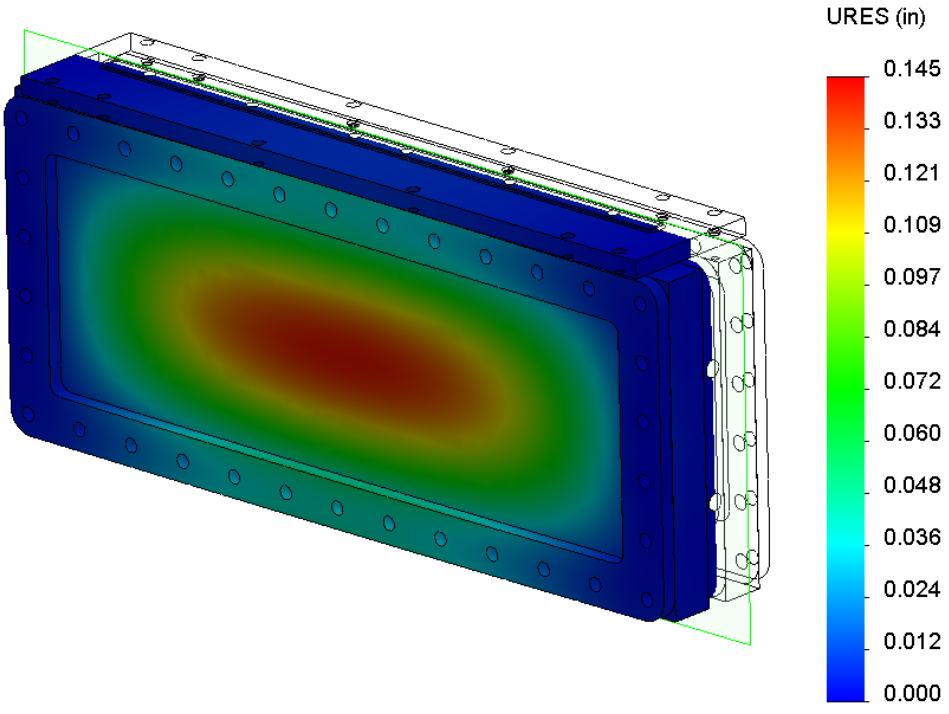


Figure A-2: Cast acrylic wall deformation at 400 psi.

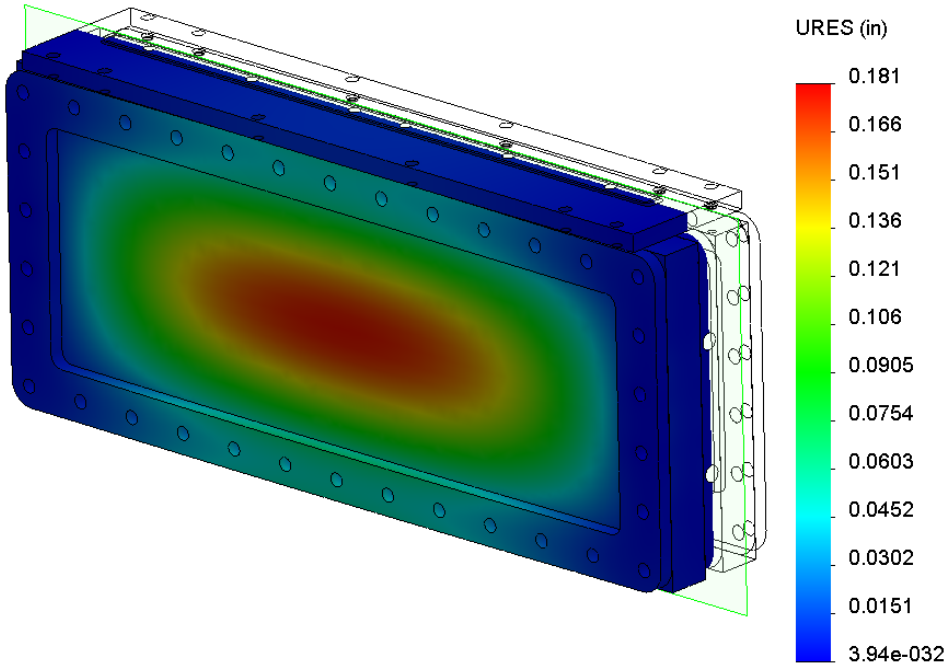


Figure A-3: Cast acrylic wall deformation at 500 psi.

A-2 Error analysis[2]

Here, the error analysis for the components of the system used to carry out experiments and obtained data is provided.

The pore volume and residual water saturation, provided in Chapters 3, 4 and 5, were obtained using the 250 ml graduated glass cylinder (70022-250, Cole-Parmer Canada Inc.) with least count of 2.0 ml. The tolerance specified by the supplier is $\pm$ 2.0 ml.

The ultimate oil recovery was measured using 15 ml test tubes (430055, Cole-Parmer Canada Inc.) with least count of 0.5 ml. As a result, it introduced the error $\pm$ 0.25 ml in measuring recovered fluid, which was accumulated during each experiment. The corresponding error bars are shown on Figures in Chapter 3, 4 and 5.

The upstream consists of dual-piston pumps coupled with mass flow meters. The flow accuracy and flow precision at 1 ml/min and 12 MPa are equal to $< 1.0\%$ and $< 0.5\%$, respectively. The mass flow meters have the measurements error equal to $\pm$ 0.2% of the full scale reading + 0.5 g/h, according to calibration certificates. The calibration has been performed by Bronkhorst Cori-Tech BV. Since, the flow rate was set at 0.5 cm^3/min, for half hour the accumulated error would be equal to $\pm$ 0.28 ml.

The densities of injected and produced liquids, provided in Chapter 3, 4 and 5, were measured using a mass flow meter (M13, Bronkhorst Cori-Tech BV). As per obtained data, the uncertainty in measurements of density for water, paraffin oil,

[2]A version of this appendix has been published as conference paper. A. Baldygin, D. S. Nobes, and S. K. Mitra, "Oil recovery from porous media using emulsion", *Proceedings of the ASME 2014 International Mechanical Engineering Congress & Exposition*, 2014

emulsion, recovered oil and produced water were $\pm$ 2.7 kg/m^3, $\pm$ 0.4 kg/m^3, $\pm$ 7.4 kg/m^3, $\pm$ 4.1 kg/m^3, $\pm$ 9.6 kg/m^3, respectively.

The viscosity of injected and produced liquids, provided in Chapter 3, 4 and 5, was measured using a rotational viscometer (Rheolab QC, Anton Paar USA Inc.) which has torque resolution of $< 0.1\mu$Nm. Hence, it would introduce an error into final value of viscosity in a range of $< 1.85 \cdot 10^{-3}\%$ without taking into account the variation of the geometry of double gap cell. As per collected data, the uncertainty in measurements of viscosity for water, paraffin oil, emulsion, recovered oil, produced water, dyed oil and dyed water were $\pm$ 0 Pa·s, $\pm$ 0.024 Pa·s, $\pm$ 16.70 Pa·s, $\pm$ 2.4 Pa·s, $\pm$ 0 Pa·s, $\pm$ 0.00018 Pa·s and $\pm$ 0 Pa·s, respectively. The viscosity of water and produced water was measured near the top limits of the double gap cell, as a result the uncertainty equals to zero.

The surface tension, provided in Chapter 4 and 5, was measured using a drop shape analyzer(DSA 100, Krüs GmbH). As per obtained data, the uncertainty in measurements of surface tension for water, paraffin oil, recovered oil, produced water, dyed oil and dyed water were $\pm$ 1.8 mN/m, $\pm$ 0.04 mN/m, $\pm$ 0.09 mN/m, $\pm$ 0.32 mN/m, $\pm$ 0.11 mN/m and $\pm$ 0.58 mN/m, respectively.

A-3 Consistency in results during 2D core flooding experiments

Here, the comparison of data from complete and incomplete 2D core flooding experiment with horizontal injection - horizontal production (HI-HP) is provided. During incomplete 2D experiment, crashing of the computer operating system has happened during process of emulsion flooding. It caused data lost for an amount of liquid injected into the system and effluent was not collected into the test tubes after 0.35 PV of emulsion injected. The main focus is in understanding the consistency in results from pack to pack for the same well configuration using visual analysis and available test tubes with collected effluent. Relevant parameters for unconsolidated sand pack samples are summarized in Table A-1.

Table A-1: Relevant parameters for unconsolidated sand pack samples used in flooding experiments.

Pack	Well configuration	Porosity, %	Pore volume, cm^3	Mass of sand pack, g	Absolute permeability, Darcy	Permeability with residual water, Darcy	$S^{\ddagger}_{wi}$
Run #10	HI-HP	34.3	820.1	4179.2	0.468 ± 0.074	0.379 ± 0.053	0.057
Run #11	HI-HP*	34.2	818.6	4183.1	0.607 ± 0.064	0.521 ± 0.032	0.03

‡ - Residual water saturation after oil saturation.
$*$ - Failed experiment

The photographs in Figure A-4 (a)-(b) show snapshots of the 2D core holder at different pore volumes of emulsion injection for HI-HP configuration. Injection wells are fully perforate with previously discussed perforation pattern. The flood front pattern has similarities in both experiments. The emulsion fills a region close to the left boundary first and then progresses towards the production well. Starting from 0.6 PV of emulsion injection, channeling is observed in Figure A-4 (b). Also, there is a flow across bottom boundary which allows to lift up oil and redirect towards production well. Overall, these images show repeatability and consistency in

experiments from pack to pack for same well configuration.

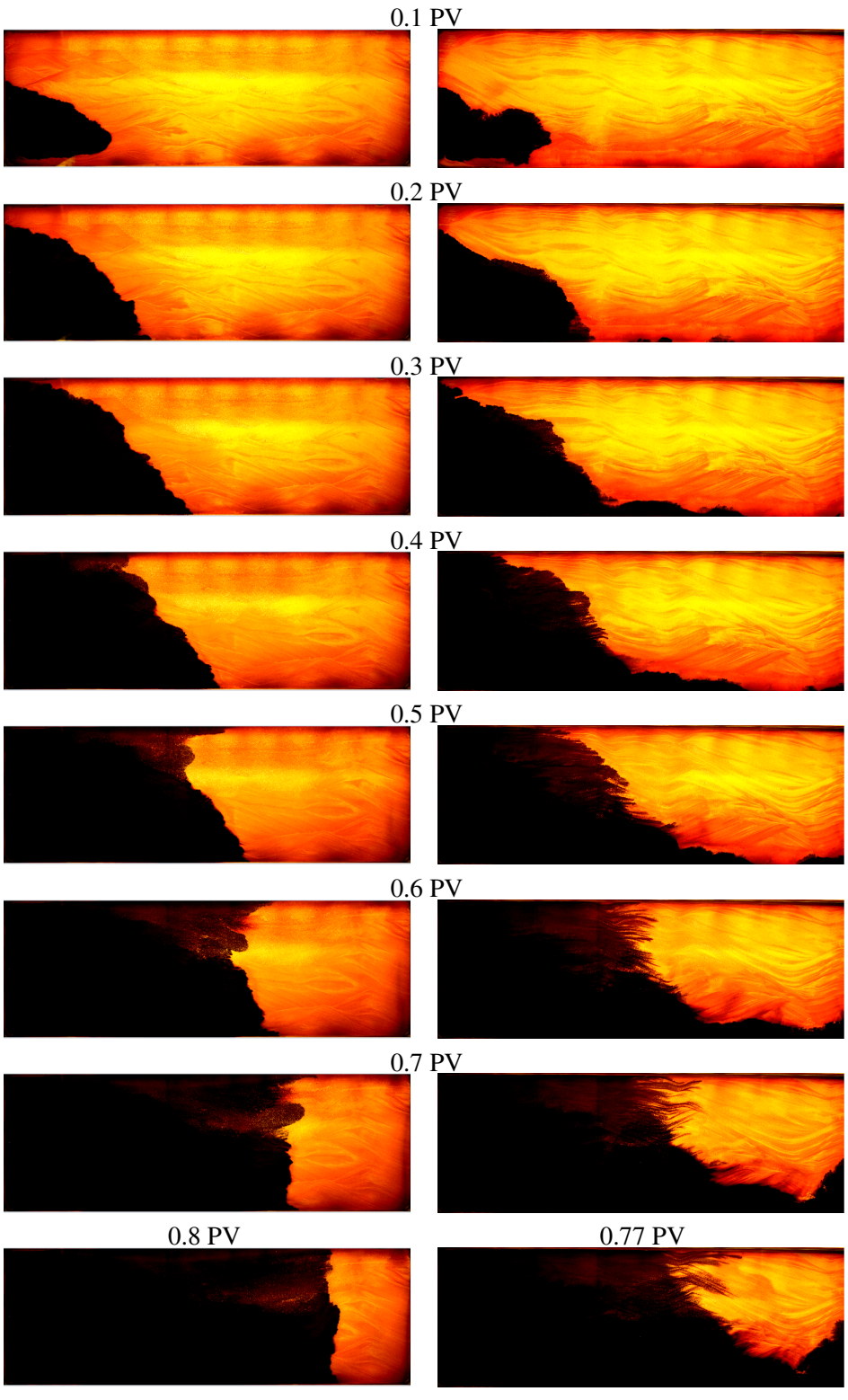


Figure A-4: Cross-section images of the sand pack for different configurations: (a) HI-HP; (b) HI-HP (failed).

Additional Run #11 was added to previously shown Figure 5.21 (a). Figure A-5 shows the ultimate oil recovery curves for three well configurations used in a 2D core holder. The recovery curve for incomplete flooding experiment follows the recovery curve for complete experiment with HI-HP well configuration. It proves repeatability and consistency in experiments from pack to pack for same well configuration.

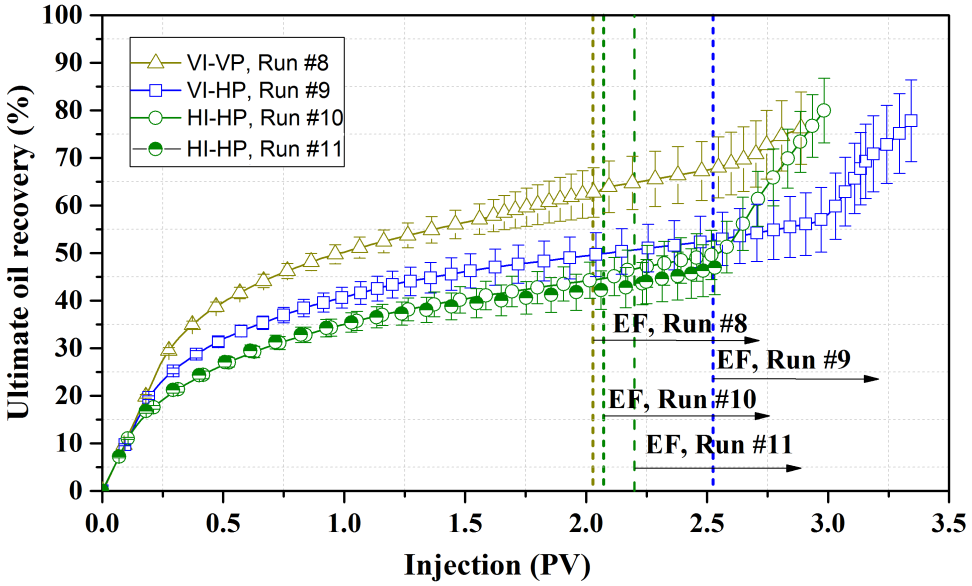


Figure A-5: Ultimate oil recovery curves for water flooding followed by emulsion flooding at different well configurations.

A-4 Image processing source code

The following code was developed using commercial multi-paradigm numerical computing environment (MATLAB R2014a, MathWorks). In order to reduce number of lines in code, due to similar procedures for each section of the core holder, the function to find profile was written into separate file, profilefinder.m. Main file, mainprogram new ref.m, recalls it for every location. Each file has detailed comments.

```
%% Finding Emulsion profile for set of images
%% Quadrise project. 2D core holder experiments
%% Aleksey Baldygin, baldygin@ualberta.ca
%% September 26, 2014

clear
clc
% File path for Run#1
image_r1 = 'Run1_complete_set/%03d.JPG';
% File path for Run#2
image_r2 = 'Run2_complete_set/%03d.JPG';
% File path for Run#3
image_r3 = 'Run3_complete_set/%03d.JPG';
%% Input parameters
%total number of images
r_r1 = 211;
r_r2 = 324;
r_r3 = 267;
        % Image resolution:
        % Run 1:
        % 3968 x 1360
        % Run 2:
        % 4884 x 1686
        % Run 3:
        % 5336 x 1792
% legth for image
l_r1 = 3968;
l_r2 = 4884;
l_r3 = 5336;
% width for image
w_r1 = 1360;
w_r2 = 1686;
w_r3 = 1792;
%(pix/mm) scale  600 - 60 cm - 600 mm
cal_r1 = 600/l_r1;
cal_r2 = 600/l_r2;
cal_r3 = 600/l_r3;
% Reference image for pore volume calculations
A_r1 = imread(sprintf(image_r1,1));
A_r2 = imread(sprintf(image_r2,1));
A_r3 = imread(sprintf(image_r3,1));
% Total area for image
area_r1 = l_r1 * w_r1;
area_r2 = l_r2 * w_r2;
area_r3 = l_r3 * w_r3;

q = 0; % how big is the slice
crop_y=100; %crop top and bottom (for removing artefacts)

crop_yb_r1 = 550;
crop_yb_r2 = 100;
```

```
crop_yb_r3 = 250;

%% Recall function to anylize data
%Run #1
[H_Z1_1,H_Z2_1,H_Z3_1,PV_1] = profilefinder(r_r1,image_r1,l_r1,w_r1,cal_r1,crop_y,↙
crop_yb_r1,q,A_r1,area_r1);
% Run #2
[H_Z1_2,H_Z2_2,H_Z3_2,PV_2] = profilefinder(r_r2,image_r2,l_r2,w_r2,cal_r2,crop_y,↙
crop_yb_r2,q,A_r2,area_r2);
% Run #3
[H_Z1_3,H_Z2_3,H_Z3_3,PV_3] = profilefinder(r_r3,image_r3,l_r3,w_r3,cal_r3,crop_y,↙
crop_yb_r3,q,A_r3,area_r3);

%% Plot, Fit and Velocity profile

fpm=1/5; % 1 image in 5 minutes
%set-up required time for complete experiment
t_r1=[0:r_r1-1]/fpm;
t_r2=[0:r_r2-1]/fpm;
t_r3=[0:r_r3-1]/fpm;

%% Run #1 // get through the data
%Re-crop matrix for future use.
% _c stands for crop
crop_H_Z1_1 = find(H_Z1_1 > 0, 1);
crop_H_Z2_1 = find(H_Z2_1 > 0, 1);
crop_H_Z3_1 = find(H_Z3_1 > 0, 1);

PV_1_1_c = PV_1 (crop_H_Z1_1:end);
PV_2_1_c = PV_1 (crop_H_Z2_1:end);
PV_3_1_c = PV_1 (crop_H_Z3_1:end);

% crop zeros at the beging
H_Z1_1_c =  H_Z1_1(crop_H_Z1_1:end);
t_Z1_1_c =  t_r1(crop_H_Z1_1:end);

H_Z2_1_c =  H_Z2_1(crop_H_Z2_1:end);
t_Z2_1_c =  t_r1(crop_H_Z2_1:end);

%H_Z3_1_c =  H_Z3_1(crop_H_Z3_1:end);
%t_Z3_1_c =  t_r1(crop_H_Z3_1:end);
% Split Bottom data in three, based on obtained results.
H_Z3_1_c_1_3 =  H_Z3_1(crop_H_Z3_1:80); % crop zeros at the beging
t_Z3_1_c_1_3 = t_r1(crop_H_Z3_1:80);
H_Z3_1_c_2_3 =  H_Z3_1(81:120); % crop zeros at the beging
t_Z3_1_c_2_3 = t_r1(81:120);
H_Z3_1_c_3_3 =  H_Z3_1(121:end); % crop zeros at the beging
t_Z3_1_c_3_3 = t_r1(121:end);

f_Z1_1=fit(t_Z1_1_c',H_Z1_1_c','poly9','Normalize','on','Robust','LAR');
f_Z2_1=fit(t_Z2_1_c',H_Z2_1_c','poly8','Normalize','on','Robust','Bisquare');
```

```
%f_Z3_1=fit(t_Z3_1_c',H_Z3_1_c','poly7','Normalize','on','Robust','Bisquare');
% Split for three curves
f_Z3_1_1_3=fit(t_Z3_1_c_1_3',H_Z3_1_c_1_3','poly9','Normalize','on','Robust','Bisquare');
f_Z3_1_2_3=fit(t_Z3_1_c_2_3',H_Z3_1_c_2_3','poly7','Normalize','on','Robust','LAR');
f_Z3_1_3_3=fit(t_Z3_1_c_3_3',H_Z3_1_c_3_3','poly7','Normalize','on','Robust','LAR');

[d1_Z1_1,d2_Z1_1] = differentiate(f_Z1_1,t_Z1_1_c); % differentiate plot
[d1_Z2_1,d2_Z2_1] = differentiate(f_Z2_1,t_Z2_1_c);
%[d1_Z3_1,d2_Z3_1] = differentiate(f_Z3_1,t_Z3_1_c);
% Split for three curves
[d1_Z3_1_1_3,d2_Z3_1_1_3] = differentiate(f_Z3_1_1_3,t_Z3_1_c_1_3);
[d1_Z3_1_2_3,d2_Z3_1_2_3] = differentiate(f_Z3_1_2_3,t_Z3_1_c_2_3);
[d1_Z3_1_3_3,d2_Z3_1_3_3] = differentiate(f_Z3_1_3_3,t_Z3_1_c_3_3);

d1_Z1_1_m_s= d1_Z1_1 * 1.66666667*10^(-5); % convert into m/s from mm/min
d1_Z2_1_m_s= d1_Z2_1 * 1.66666667*10^(-5);
% d1_Z3_1_m_s= d1_Z3_1 * 1.66666667*10^(-5);

% Split for 3 curves
d1_Z3_1_m_s_1_3= d1_Z3_1_1_3 * 1.66666667*10^(-5);
d1_Z3_1_m_s_2_3= d1_Z3_1_2_3 * 1.66666667*10^(-5);
d1_Z3_1_m_s_3_3= d1_Z3_1_3_3 * 1.66666667*10^(-5);
% combine back
d1_Z3_1_m_s=  cat(1, d1_Z3_1_m_s_1_3, d1_Z3_1_m_s_2_3, d1_Z3_1_m_s_3_3);
t_Z3_1_c = cat (2, t_Z3_1_c_1_3, t_Z3_1_c_2_3, t_Z3_1_c_3_3);

%% Run #2 // get through the data
%Re-crop matrix for future use.
% _c stands for crop
crop_H_Z1_2 = find(H_Z1_2 > 0, 1);
% crop_H_Z2_2 = find(H_Z2_2 > 0, 1);
crop_H_Z3_2 = find(H_Z3_2 > 0, 1);

PV_1_2_c = PV_2 (crop_H_Z1_2:end);
%PV_2_2_c = PV_2 (crop_H_Z2_2:end);
PV_3_2_c = PV_2 (crop_H_Z3_2:end);


H_Z1_2_c =  H_Z1_2(crop_H_Z1_2:end); % crop zeros at the beging
 t_Z1_2_c = t_r2(crop_H_Z1_2:end);

%H_Z3_2_c =  H_Z3_2(crop_H_Z3_2:end); % crop zeros at the beging
%t_Z3_2_c = t_r2(crop_H_Z3_2:end);

% Split Bottom data in two, based on obtained results.
H_Z3_2_c_1_2 =  H_Z3_2(crop_H_Z3_2:120); % crop zeros at the beging
t_Z3_2_c_1_2 = t_r2(crop_H_Z3_2:120);
H_Z3_2_c_2_2 =  H_Z3_2(121:end); % crop zeros at the beging
t_Z3_2_c_2_2 = t_r2(121:end);
```

```
f_Z1_2=fit(t_Z1_2_c',H_Z1_2_c','poly9','Normalize','on','Robust','Bisquare');
f_Z2_2=fit(t_r2',H_Z2_2','poly7','Normalize','on','Robust','Bisquare');

%f_Z3_2=fit(t_Z3_2_c',H_Z3_2_c','poly8','Normalize','on','Robust','Bisquare');
% Split for two curves
f_Z3_2_1_2=fit(t_Z3_2_c_1_2',H_Z3_2_c_1_2','poly6','Normalize','on','Robust','Bisquare');
f_Z3_2_2_2=fit(t_Z3_2_c_2_2',H_Z3_2_c_2_2','poly7','Normalize','on','Robust','Bisquare');

[d1_Z1_2,d2_Z1_2] = differentiate(f_Z1_2,t_Z1_2_c); % differentiate plot
[d1_Z2_2,d2_Z2_2] = differentiate(f_Z2_2,t_r2);
%[d1_Z3_2,d2_Z3_2] = differentiate(f_Z3_2,t_r2);
% Split for two curves
[d1_Z3_2_1_2,d2_Z3_2_1_2] = differentiate(f_Z3_2_1_2,t_Z3_2_c_1_2);
[d1_Z3_2_2_2,d2_Z3_2_2_2] = differentiate(f_Z3_2_2_2,t_Z3_2_c_2_2);


d1_Z1_2_m_s= d1_Z1_2 * 1.66666667*10^(-5); % convert into m/s from mm/min
d1_Z2_2_m_s= d1_Z2_2 * 1.66666667*10^(-5);
%d1_Z3_2_m_s= d1_Z3_2 * 1.66666667*10^(-5);
% Split for 2 curves
d1_Z3_2_m_s_1_2= d1_Z3_2_1_2 * 1.66666667*10^(-5);
d1_Z3_2_m_s_2_2= d1_Z3_2_2_2 * 1.66666667*10^(-5);
% combine back
d1_Z3_2_m_s_com =  cat(1, d1_Z3_2_m_s_1_2, d1_Z3_2_m_s_2_2);
t_Z3_2_c_com = cat (2, t_Z3_2_c_1_2, t_Z3_2_c_2_2);

%% Run #3 // get through the data
%Re-crop matrix for future use.
% _c stands for crop
crop_H_Z1_3 = find(H_Z1_3 > 0, 1);
crop_H_Z2_3 = find(H_Z2_3 > 0, 1);
crop_H_Z3_3 = find(H_Z3_3 > 0, 1);

PV_1_3_c = PV_3 (crop_H_Z1_3:end);
PV_2_3_c = PV_3 (crop_H_Z2_3:end);
PV_3_3_c = PV_3 (crop_H_Z3_3:end);


H_Z1_3_c =  H_Z1_3(crop_H_Z1_3:end); % crop zeros at the beging
 t_Z1_3_c = t_r3(crop_H_Z1_3:end);

H_Z2_3_c =  H_Z2_3(crop_H_Z2_3:end); % crop zeros at the beging
 t_Z2_3_c = t_r3(crop_H_Z2_3:end);

H_Z3_3_c =  H_Z3_3(crop_H_Z3_3:end); % crop zeros at the beging
 t_Z3_3_c = t_r3(crop_H_Z3_3:end);


f_Z1_3=fit(t_Z1_3_c',H_Z1_3_c','poly9','Normalize','on','Robust','Bisquare');
f_Z2_3=fit(t_Z2_3_c',H_Z2_3_c','poly7','Normalize','on','Robust','Bisquare');
f_Z3_3=fit(t_Z3_3_c',H_Z3_3_c','poly6','Normalize','on','Robust','Bisquare');
```

```
[d1_Z1_3,d2_Z1_3] = differentiate(f_Z1_3,t_Z1_3_c); % differentiate plot
[d1_Z2_3,d2_Z2_3] = differentiate(f_Z2_3,t_Z2_3_c);
[d1_Z3_3,d2_Z3_3] = differentiate(f_Z3_3,t_Z3_3_c);

d1_Z1_3_m_s= d1_Z1_3 * 1.66666667*10^(-5); % convert into m/s from mm/min
d1_Z2_3_m_s= d1_Z2_3 * 1.66666667*10^(-5);
d1_Z3_3_m_s= d1_Z3_3 * 1.66666667*10^(-5);

%% Plot Run %1
scrsz = get(0,'ScreenSize'); % get screen size
figure('Position',[0 scrsz(4)/4 1000 500])
% figure('Position', [x y width height])
plot(t_Z1_1_c,d1_Z1_1_m_s,'r',t_Z2_1_c,d1_Z2_1_m_s,'g',t_Z3_1_c,d1_Z3_1_m_s,'c',t_r1,↙
7.309*10^(-6),'b') % double plot method
title('Velocity')
axis([0,max(t_r1),0,1.3*10^(-5)])
%axis normal
set(gca,'YTick',[10^(-6),2*10^(-6),4*10^(-6),6*10^(-6),8*10^(-6),1*10^(-5),1.25*10^(-5)])
xlabel('time, [min]')
ylabel('speed, [m/s]')
grid on
hleg1 = legend('top','center','bottom','reference');

fig1 = figure('Position',[1000 0 1000 1500])
subplot(3,1,1)
plot(f_Z1_1, t_r1', H_Z1_1','k.')
title('Top','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp1 = legend('Front location','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r1),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,2)
plot(f_Z2_1, t_r1', H_Z2_1','k.')
title('Center','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp2 = legend('Front location','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r1),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,3)
hold on
box on
plot (f_Z3_1_1_3, t_Z3_1_c_1_3', H_Z3_1_c_1_3','k.')
plot (f_Z3_1_2_3, t_Z3_1_c_2_3', H_Z3_1_c_2_3','k.')
plot (f_Z3_1_3_3, t_Z3_1_c_3_3', H_Z3_1_c_3_3','k.')
```

```
% plot(f_Z3_1, t_r1', H_Z3_1')
title('Bottom','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp3 = legend('Front location','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r1),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
set(gcf,'PaperPositionMode','auto'); % freeze configuration for image, to save image as
it is shown on the screen
print(fig1,'-dpng','VIVP');
print(fig1,'-depsc','VIVP');



%% Plot Run # 2
scrsz = get(0,'ScreenSize'); % get screen size
figure('Position',[0 scrsz(4)/4 1000 500])
% figure('Position', [x y width height])
hold on
plot(t_Z1_2_c,d1_Z1_2_m_s,'r',t_r2,d1_Z2_2_m_s,'g',t_Z3_2_c_1_2,d1_Z3_2_m_s_1_2,'c',t_r2,
7.309*10^(-6),'b') % double plot method
plot(t_Z3_2_c_2_2,d1_Z3_2_m_s_2_2,'c')
title('Velocity')
axis([0,max(t_r2),0,2.5*10^(-5)])
%axis normal
set(gca,'YTick',[10^(-6),2*10^(-6),4*10^(-6),6*10^(-6),8*10^(-6),1*10^(-5),1.25*10^(-5)])
xlabel('time, [min]')
ylabel('speed, [m/s]')
grid on
hleg1 = legend('top','center','bottom','reference');

fig2 = figure('Position',[1000 0 1000 1500])
subplot(3,1,1)
plot(f_Z1_2, t_r2', H_Z1_2','k.')
title('Top','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp1 = legend('VI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r2),0,600]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,2)
plot(f_Z2_2, t_r2', H_Z2_2','k.')
title('Center','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp2 = legend('VI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
```

```
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r3),0,600]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,3)
hold on
box on
plot(f_Z3_2_1_2, t_Z3_2_c_1_2', H_Z3_2_c_1_2','k.')
plot(f_Z3_2_2_2, t_Z3_2_c_2_2', H_Z3_2_c_2_2','k.')
title('Bottom','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp3 = legend('VI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r2),0,600]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
set(gcf,'PaperPositionMode','auto'); % freeze configuration for image, to save image as↙
it is shown on the screen
print(fig2,'-dpng','VIHP');
print(fig2,'-depsc','VIHP');



%% Plot Run # 3
scrsz = get(0,'ScreenSize'); % get screen size
figure('Position',[0 scrsz(4)/4 1000 500])
% figure('Position', [x y width height])
plot(t_Z1_3_c,d1_Z1_3_m_s,'r',t_Z2_3_c,d1_Z2_3_m_s,'g',t_Z3_3_c,d1_Z3_3_m_s,'c',t_r3,↙
7.309*10^(-6),'b') % double plot method
title('Velocity')
axis([0,max(t_r3),0,2.25*10^(-5)])
%axis normal
set(gca,'YTick',[10^(-6),2*10^(-6),4*10^(-6),6*10^(-6),8*10^(-6),1*10^(-5),1.25*10^(-5)])
xlabel('time, [min]')
ylabel('speed, [m/s]')
grid on
hleg1 = legend('top','center','bottom','reference');

fig3 = figure('Position',[1000 0 1000 1500])
subplot(3,1,1)
plot(f_Z1_3, t_Z1_3_c', H_Z1_3_c','k.')
title('Top','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp1 = legend('HI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r3),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,2)
```

```
plot(f_Z2_3, t_Z2_3_c', H_Z2_3_c','k.')
title('Center','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp2 = legend('HI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r3),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,3)
plot(f_Z3_3, t_Z3_3_c', H_Z3_3_c','k.')
title('Bottom','FontWeight','bold','FontSize',14,'FontName','Arial')
hleg_sp3 = legend('HI-HP','Poly-fit','Location','SouthEast');
ylabel('position, [mm]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('time, [min]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
axis([0,max(t_r3),0,600])
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
set(gcf,'PaperPositionMode','auto'); % freeze configuration for image, to save image as↙
it is shown on the screen
print(fig3,'-dpng','HIHP');
print(fig3,'-depsc','HIHP');

%% Combined Runs into single plot vs Time % Final for paper
figure('Position',[1000 0 1000 1500])
subplot(3,1,1)
hold on
% Run #1
plot (t_Z1_1_c,d1_Z1_1_m_s,'k') % top
% Run #2
plot(t_Z1_2_c,d1_Z1_2_m_s,'g') % top
% Run #3
plot(t_Z1_3_c,d1_Z1_3_m_s,'r') % top
plot(t_r3,7.309*10^(-6),'b') % reference
title('Top')
hleg_sp1 = legend('VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('position, [mm]')
xlabel('time, [min]')
grid on
axis([0,max(t_r3),0,1.5*10^(-5)])
subplot(3,1,2)
hold on
% Run #1
plot(t_Z2_1_c,d1_Z2_1_m_s,'k') % Center
% Run #2
plot(t_r2,d1_Z2_2_m_s,'g') % Center
% Run #3
plot(t_Z2_3_c,d1_Z2_3_m_s,'r')  % Center
plot(t_r3,7.309*10^(-6),'b') % reference
title('Center')
```

```
hleg_sp2 = legend('VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('position, [mm]')
xlabel('time, [min]')
grid on
axis([0,max(t_r3),0,1.5*10^(-5)])
subplot(3,1,3)
hold on
% Run #1
plot (t_Z3_1_c,d1_Z3_1_m_s,'k')
% Run #2
%h1_1_2= plot(t_Z3_2_c_1_2,d1_Z3_2_m_s_1_2,'g',t_Z3_2_c_2_2,d1_Z3_2_m_s_2_2,'g')
plot (t_Z3_2_c_com,d1_Z3_2_m_s_com,'g')
% Run #3
plot(t_Z3_3_c,d1_Z3_3_m_s,'r')  % bottom
plot(t_r3,7.309*10^(-6),'b-') % reference
title('Bottom')
legend('VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('position, [mm]')
xlabel('time, [min]')
grid on
axis([0,max(t_r3),0,1.5*10^(-5)])


%% Combined Runs into single plot vs PV % Final for paper
fig4 = figure('Position',[1000 0 1000 1500])
subplot(3,1,1)
hold on
% Run #1
plot (PV_1_1_c,d1_Z1_1_m_s,'k','LineWidth',1.5) % top
% Run #2
plot (PV_1_2_c,d1_Z1_2_m_s,'g','LineWidth',1.5) % top
% Run #3
plot (PV_1_3_c,d1_Z1_3_m_s,'r','LineWidth',1.5) % top
plot(PV_1,1.177*10^(-5),'k','LineWidth',2.5) % reference
plot(PV_1,1.204*10^(-5),'g','LineWidth',2.5) % reference
plot(PV_1,1.215 *10^(-5),'r','LineWidth',2.5) % reference
title('Top','FontWeight','bold','FontSize',14,'FontName','Arial');
legend('VI-VP','VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('Velocity, [m/s]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('Emulsion Injected, [PV]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
grid on
axis([0,1,0,1.5*10^(-5)]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,2)
hold on
% Run #1
plot (PV_2_1_c,d1_Z2_1_m_s,'k','LineWidth',1.5) % Center
% Run #2
plot (PV_2,d1_Z2_2_m_s,'g','LineWidth',1.5) % Center
```

```
% Run #3
plot (PV_2_3_c,d1_Z2_3_m_s,'r','LineWidth',1.5)  % Center
plot(PV_2,1.177*10^(-5),'k','LineWidth',2.5) % reference
plot(PV_2,1.204*10^(-5),'g','LineWidth',2.5) % reference
plot(PV_2,1.215 *10^(-5),'r','LineWidth',2.5) % reference
title('Center','FontWeight','bold','FontSize',14,'FontName','Arial');
legend('VI-VP','VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('Velocity, [m/s]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('Emulsion Injected, [PV]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
grid on
axis([0,1,0,1.5*10^(-5)]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
subplot(3,1,3)
hold on
% Run #1
plot (PV_3_1_c,d1_Z3_1_m_s,'k','LineWidth',1.5)
% Run #2
%h1_1_2= plot(t_Z3_2_c_1_2,d1_Z3_2_m_s_1_2,'g',t_Z3_2_c_2_2,d1_Z3_2_m_s_2_2,'g')
plot (PV_3_2_c,d1_Z3_2_m_s_com,'g','LineWidth',1.5)
% Run #3
plot (PV_3_3_c,d1_Z3_3_m_s,'r','LineWidth',1.5)  % bottom
plot(PV_3,1.177*10^(-5),'k','LineWidth',2.5) % reference
plot(PV_3,1.204*10^(-5),'g','LineWidth',2.5) % reference
plot(PV_3,1.215 *10^(-5),'r','LineWidth',2.5) % reference
title('Bottom','FontWeight','bold','FontSize',14,'FontName','Arial');
legend('VI-VP','VI-HP','HI-HP','Reference','Location','NorthEast');
ylabel('Velocity, [m/s]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
xlabel('Emulsion Injected, [PV]','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
grid on
axis([0,1,0,1.5*10^(-5)]);
set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
set(gcf,'PaperPositionMode','auto'); % freeze configuration for image, to save image as↙
it is shown on the screen
print(fig4,'-dpng','Figure10');
print(fig4,'-depsc','Figure10');
```

```
function [H_Z1,H_Z2,H_Z3,PV] = profilefinder(r,image,l,w,cal,crop_y,crop_yb,q,A_ref,ref)

%prepare space for data (front)
H_Z1=zeros(1,r);
H_Z2=zeros(1,r);
H_Z3=zeros(1,r);

%Set locations for speed profiles:
w_3=round(w/4); %bottom
w_2=2*w_3;      %center
w_1=3*w_3;      %top

for k=1:r

%Read nth image!
    A = imread(sprintf(image,k)); %read out images
    B = imrotate(A,270); %270 degrees, since 'edge'-function starts from the top of the↙
image

% First position
Z1=B(crop_y:l-crop_yb,w_1-q:w_1+q,1,:);
Z1_bw=im2bw(Z1,100/255);
Z1_edge=edge(Z1_bw,'prewitt');
find_loc_Z1 = find(Z1_edge == 1);
    if find_loc_Z1 > 0 % if emulsion was detected + correct location for crop value
    location_corr_Z1 = find(Z1_edge == 1) + crop_y - 1;
    else
    location_corr_Z1 = 0;
    end

y_Z1 = mean(location_corr_Z1); % Most frequent values in array
H_Z1(k) = y_Z1*cal; % TOP
% Second position
    Z2=B(crop_y:l-crop_yb,w_2-q:w_2+q,1,:); %red channel
    Z2_bw=im2bw(Z2,100/255);
    Z2_edge=edge(Z2_bw,'prewitt');
    find_loc_Z2 = find(Z2_edge == 1); % search for front location
    if find_loc_Z2 > 0 % if emulsion was detected + correct location for crop value
    location_corr_Z2 = find(Z2_edge == 1) + crop_y - 1;
    else            % if there is no emulsion, set to zero
    location_corr_Z2 = 0;
    end

    y_Z2 = max(location_corr_Z2); % set location
    H_Z2(k)=y_Z2*cal; % correct location with calibration (\approx number)

% Third position
Z3=B(crop_y:l-crop_yb,w_3-q:w_3+q,1,:);
Z3_bw=im2bw(Z3,100/255);
Z3_edge=edge(Z3_bw,'prewitt');
find_loc_Z3 = find(Z3_edge == 1);
```

```
    if find_loc_Z3 > 0 % if emulsion was detected + correct location for crop value
    location_corr_Z3 = find(Z3_edge == 1) + crop_y - 1;
    elseif isempty(find_loc_Z3) && (k > 100) % if there is no emulsion, set to zero
        % do nothing
    else % if there is no emulsion, set to zero
    location_corr_Z3 = 0;
    end

y_Z3 = max(location_corr_Z3);
H_Z3(k)=y_Z3*cal; % BOTTOM

%% Finding PV

A_an = imsubtract (A_ref, A);
A_an_RED = A_an(:,:,1,:);
A_an_RED_bw = im2bw(A_an_RED,40/255);
total = bwarea(A_an_RED_bw);
PV(k) = 1 - (ref - total) / ref;

end

return
```

A-5 Pressure data processing source code

The following code was developed using commercial multi-paradigm numerical computing environment (MATLAB R2014a, MathWorks) to process data collected from 15 pressure sensors during 2D core flooding studies. Each file has detailed comments.

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    Run3_emulsion_data.log
%
%
%
clear
clc


%% Initialize variables.
filename = 'Run1_emulsion.log';
delimiter = '\t';

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%*s%*s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%↙
*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%↙
*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,  'ReturnOnError',↙
false);

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+]↙
*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
```

```
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
Timeinmin = cell2mat(raw(:, 1));
PoreVolumePV = cell2mat(raw(:, 2));
Pressureinlet = cell2mat(raw(:, 3));
EMULSIONIN = cell2mat(raw(:, 4));
CP11 = cell2mat(raw(:, 5));
CP12 = cell2mat(raw(:, 6));
CP13 = cell2mat(raw(:, 7));
CP14 = cell2mat(raw(:, 8));
CP15 = cell2mat(raw(:, 9));
CP21 = cell2mat(raw(:, 10));
CP22 = cell2mat(raw(:, 11));
CP23 = cell2mat(raw(:, 12));
CP24 = cell2mat(raw(:, 13));
CP25 = cell2mat(raw(:, 14));
CP31 = cell2mat(raw(:, 15));
CP32 = cell2mat(raw(:, 16));
CP33 = cell2mat(raw(:, 17));
CP34 = cell2mat(raw(:, 18));
CP35 = cell2mat(raw(:, 19));
```

```
EMULSIONIN (1) = 0;
PoreVolumePV(1) = PoreVolumePV(2);
Pressureinlet(1) = Pressureinlet(2);
Timeinmin (1) = 0;

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans raw col numericData rawData↙
row regexstr result numbers invalidThousandsSeparator thousandsRegExp me R;

CP11_f = zeros (size(CP11,1),1);
CP12_f = zeros (size(CP12,1),1);
CP13_f = zeros (size(CP13,1),1);
CP14_f = zeros (size(CP14,1),1);
CP15_f = zeros (size(CP15,1),1);
CP21_f = zeros (size(CP21,1),1);
CP22_f = zeros (size(CP22,1),1);
CP23_f = zeros (size(CP23,1),1);
CP24_f = zeros (size(CP24,1),1);
CP25_f = zeros (size(CP25,1),1);
CP31_f = zeros (size(CP31,1),1);
CP32_f = zeros (size(CP32,1),1);
CP33_f = zeros (size(CP33,1),1);
CP34_f = zeros (size(CP34,1),1);
CP35_f = zeros (size(CP35,1),1);

CP11_f(:) = - CP11(:);
CP12_f(:) = - CP12(:);
CP13_f(:) = - CP13(:);
CP14_f(:) =  CP14(:);
CP15_f(:) =  CP15(:);
CP21_f(:) =  CP21(:);
CP22_f(:) = - CP22(:);
CP23_f(:) =  CP23(:);
CP24_f(:) =  CP24(:);
CP25_f(:) = - CP25(:);
CP31_f(:) = - CP31(:);
CP32_f(:) = - CP32(:);
CP33_f(:) = - CP33(:);
CP34_f(:) =  CP34(:);
CP35_f(:) =  CP35(:);

clearvars CP11 CP12 CP13 CP14 CP15 CP21 CP22 CP23 CP24 CP25 CP31 CP32 CP33 CP34 CP35;

% load data from image analysis
load('PVandTime_from_images.mat');
clearvars PV_2 PV_3 t_r2 t_r3;


% temp time for double check
temp_time = zeros (size(t_r1,2),1);
```

```
%% set coordinates for surface plot
x_crd = zeros (15,1);
x_crd(1) = 50;
x_crd(2) = 175;
x_crd(3) = 300;
x_crd(4) = 425;
x_crd(5) = 550;

x_crd(6) = 50;
x_crd(7) = 175;
x_crd(8) = 300;
x_crd(9) = 425;
x_crd(10) = 550;

x_crd(11) = 50;
x_crd(12) = 175;
x_crd(13) = 300;
x_crd(14) = 425;
x_crd(15) = 550;

y_crd = zeros (15,1);
y_crd(1) = 160;
y_crd(2) = 160;
y_crd(3) = 160;
y_crd(4) = 160;
y_crd(5) = 160;

y_crd(6) = 100;
y_crd(7) = 100;
y_crd(8) = 100;
y_crd(9) = 100;
y_crd(10) = 100;

y_crd(11) = 40;
y_crd(12) = 40;
y_crd(13) = 40;
y_crd(14) = 40;
y_crd(15) = 40;

CP = zeros (15,size(t_r1,2));
%%
step_image = 2;
for step = 1 : size (Timeinmin,1)
    if (Timeinmin(step) >=t_r1(step_image))
        % save current time for further reference
        temp_time (step_image) = Timeinmin(step,1);

        %Set new step for pressure data
        step_p_min = step - 10;
        step_p_max = step + 10;
```

```
        %Trim new data for plotting +/- 10 steps from given point
        %1st row
        CP11_f_temp = CP11_f (step_p_min:step_p_max);
        CP (1,step_image) = mean2(CP11_f_temp);
        CP12_f_temp = CP12_f (step_p_min:step_p_max);
        CP (2,step_image) = mean2(CP12_f_temp);
        CP13_f_temp = CP13_f (step_p_min:step_p_max);
        CP (3,step_image) = mean2(CP13_f_temp);
        CP14_f_temp = CP14_f (step_p_min:step_p_max);
        CP (4,step_image) = mean2(CP14_f_temp);
        CP15_f_temp = CP15_f (step_p_min:step_p_max);
        CP (5,step_image) = mean2(CP15_f_temp);

        %2nd row
        CP21_f_temp = CP21_f (step_p_min:step_p_max);
        CP (6,step_image) = mean2(CP21_f_temp);
        CP22_f_temp = CP22_f (step_p_min:step_p_max);
        CP (7,step_image) = mean2(CP22_f_temp);
        CP23_f_temp = CP23_f (step_p_min:step_p_max);
        CP (8,step_image) = mean2(CP23_f_temp);
        CP24_f_temp = CP24_f (step_p_min:step_p_max);
        CP (9,step_image) = mean2(CP24_f_temp);
        CP25_f_temp = CP25_f (step_p_min:step_p_max);
        CP (10,step_image) = mean2(CP25_f_temp);

        %3rd row
        CP31_f_temp = CP31_f (step_p_min:step_p_max);
        CP (11,step_image) = mean2(CP31_f_temp);
        CP32_f_temp = CP32_f (step_p_min:step_p_max);
        CP (12,step_image) = mean2(CP32_f_temp);
        CP33_f_temp = CP33_f (step_p_min:step_p_max);
        CP (13,step_image) = mean2(CP33_f_temp);
        CP34_f_temp = CP34_f (step_p_min:step_p_max);
        CP (14,step_image) = mean2(CP34_f_temp);
        CP35_f_temp = CP35_f (step_p_min:step_p_max);
        CP (15,step_image) = mean2(CP35_f_temp);

        % next point for time-data
        step_image = step_image + 1;
        if (step_image > size(t_r1,2))
            break;
        end
    end
end

% Load background image
img = imread('run1background.png');
min_x = 0;
max_x = 600;
min_y = 0;
max_y = 200;
```

```
fig1 = figure('Position',[500 100 1200 350],'Color',[1 1 1]); %500 left coord. 100 bottom coord. 1200 width 400 height
winsize = get(fig1,'Position'); %set position, dimension of figure for future reference.
winsize(1:2) = [0 0]; %adjust windows size for recording
numframes=size(CP,2); %determines number of frames
set(fig1,'nextplot','replacechildren');
set(gcf,'Renderer','zbuffer');

image = 'Run1/Run1_%03d'; %set name for saving .png-files
        pvinj = 'Emulsion inj. = %1.4f PV'; %set name for legends

writerObj = VideoWriter('run1.avi'); %prepares .avi-file for saving frames
open(writerObj); %opens .avi-file for saving

for i=1:numframes
    % Get current data for plot
    z_p=CP(:,i);
    Data = scatteredInterpolant(x_crd,y_crd,z_p); %converts data for sutable format
    x_n = 50:10:550;
    y_n = 40:10:160;
    [x_nn,y_nn] = meshgrid(x_n,y_n);
    z_nn = Data (x_nn,y_nn);
    hold off;
    imagesc([min_x max_x], [min_y max_y], flipud(img));
    hold on;
    contourf(x_nn,y_nn,z_nn,50); % plot countor plot

    %set(h,'EdgeColor','none');
    % grid on
    set(gca,'ydir','normal'); % restore normal axis after loading background

        % Settings for labes, colorbar, axis, title and legend:
        axis([0,600,0,200]) % set axis
        caxis([0, 10])
        set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create xlabel
        xlabel('x coodrinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create ylabel
        ylabel('y coordinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create colorbar
        colorbar('FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create title
        title('Pressure map','FontWeight','bold','FontSize',14,'FontName','Arial');
        legend(sprintf(pvinj ,PV_1(i)),'Location','SouthEast');
        legend('boxoff');
    frame = getframe(fig1,winsize); %get window size
    writeVideo(writerObj,frame);    %write figure in avi-file
    %saveas(fig1,sprintf(image,i),'fig') %save a matlab-fig file
```

```
    set(gcf,'PaperPositionMode','auto') % freeze configuration for image, to save image↙
as it is shown on the screen
    print(fig1,'-dpng',sprintf(image,i))
end
close(writerObj);


% References:
% 1. Videowriter
% http://www.mathworks.com/help/matlab/ref/videowriter-class.html
% 2. Creating movies in MATLAB
% http://www.math.canterbury.ac.nz/~c.scarrott/MATLAB_Movies/movies.html
% 3. Background image
% http://www.peteryu.ca/tutorials/matlab/plot_over_image_background
% 4. Save image as png-file
% http://www.mathworks.com/help/matlab/ref/print.html?searchHighlight=print
```

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    Run2_emulsion_running_time_in_min.log
%
%
%

%% Initialize variables.
filename = 'Run2_emulsion_running_time_in_min.log';
delimiter = '\t';

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%*s%*s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s% ↙
*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s% ↙
*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,  'ReturnOnError', ↙
false);

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+] ↙
*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;
```

```
            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
Timeinmin = cell2mat(raw(:, 1));
PoreVolumePV = cell2mat(raw(:, 2));
Pressureinlet = cell2mat(raw(:, 3));
EMULSIONIN = cell2mat(raw(:, 4));
CP11 = cell2mat(raw(:, 5));
CP12 = cell2mat(raw(:, 6));
CP13 = cell2mat(raw(:, 7));
CP14 = cell2mat(raw(:, 8));
CP15 = cell2mat(raw(:, 9));
CP21 = cell2mat(raw(:, 10));
CP22 = cell2mat(raw(:, 11));
CP23 = cell2mat(raw(:, 12));
CP24 = cell2mat(raw(:, 13));
CP25 = cell2mat(raw(:, 14));
CP31 = cell2mat(raw(:, 15));
CP32 = cell2mat(raw(:, 16));
CP33 = cell2mat(raw(:, 17));
CP34 = cell2mat(raw(:, 18));
CP35 = cell2mat(raw(:, 19));

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans raw col numericData rawData↙
row regexstr result numbers invalidThousandsSeparator thousandsRegExp me R;
```

```
CP11_f = zeros (size(CP11,1),1);
CP12_f = zeros (size(CP12,1),1);
CP13_f = zeros (size(CP13,1),1);
CP14_f = zeros (size(CP14,1),1);
CP15_f = zeros (size(CP15,1),1);
CP21_f = zeros (size(CP21,1),1);
CP22_f = zeros (size(CP22,1),1);
CP23_f = zeros (size(CP23,1),1);
CP24_f = zeros (size(CP24,1),1);
CP25_f = zeros (size(CP25,1),1);
CP31_f = zeros (size(CP31,1),1);
CP32_f = zeros (size(CP32,1),1);
CP33_f = zeros (size(CP33,1),1);
CP34_f = zeros (size(CP34,1),1);
CP35_f = zeros (size(CP35,1),1);

CP11_f(:) = - CP11(:);
CP12_f(:) = - CP12(:);
CP13_f(:) = - CP13(:);
CP14_f(:) = - CP14(:);
CP15_f(:) = - CP15(:);
CP21_f(:) = - CP21(:);
CP22_f(:) = - CP22(:);
CP23_f(:) = - CP23(:);
CP24_f(:) = - CP24(:);
CP25_f(:) = - CP25(:);
CP31_f(:) = - CP31(:);
CP32_f(:) = - CP32(:);
CP33_f(:) = - CP33(:);
CP34_f(:) = - CP34(:);
CP35_f(:) = - CP35(:);

clearvars CP11 CP12 CP13 CP14 CP15 CP21 CP22 CP23 CP24 CP25 CP31 CP32 CP33 CP34 CP35;

% load data from image analysis
load('PVandTime_from_images.mat');

% temp time for double check
temp_time = zeros (size(t_r2,2),1);

%% set coordinates for surface plot
x_crd = zeros (10,1);
x_crd(1) = 50;
x_crd(2) = 175;
x_crd(3) = 300;
%x_crd(4) = 425;
%x_crd(5) = 550;

%x_crd(6) = 50;
x_crd(4) = 175;
x_crd(5) = 300;
```

```
%x_crd(9) = 425;
x_crd(6) = 550;

x_crd(7) = 50;
x_crd(8) = 175;
x_crd(9) = 300;
x_crd(10) = 425;
%x_crd(15) = 550;

y_crd = zeros (10,1);
y_crd(1) = 160;
y_crd(2) = 160;
y_crd(3) = 160;
%y_crd(4) = 160;
%y_crd(5) = 160;

%y_crd(6) = 100;
y_crd(4) = 100;
y_crd(5) = 100;
%y_crd(9) = 100;
y_crd(6) = 100;

y_crd(7) = 40;
y_crd(8) = 40;
y_crd(9) = 40;
y_crd(10) = 40;
%y_crd(15) = 40;

CP = zeros (10,size(t_r2,2));
%%
step_image = 2;
for step = 1 : size (Timeinmin,1)
    if (Timeinmin(step) >=t_r2(step_image))
        % save current time for further reference
        temp_time (step_image) = Timeinmin(step,1);

        %Set new step for pressure data
        step_p_min = step - 10;
        step_p_max = step + 10;

        %Trim new data for plotting +/- 10 steps from given point
        %1st row
        CP11_f_temp = CP11_f (step_p_min:step_p_max);
        CP (1,step_image) = mean2(CP11_f_temp);
        CP12_f_temp = CP12_f (step_p_min:step_p_max);
        CP (2,step_image) = mean2(CP12_f_temp);
        CP13_f_temp = CP13_f (step_p_min:step_p_max);
        CP (3,step_image) = mean2(CP13_f_temp);
        %CP14_f_temp = CP14_f (step_p_min:step_p_max);
        %CP (4,step_image) = mean2(CP14_f_temp);
        %CP15_f_temp = CP15_f (step_p_min:step_p_max);
```

```
        %CP (5,step_image) = mean2(CP15_f_temp);

        %2nd row
        %CP21_f_temp = CP21_f (step_p_min:step_p_max);
        %CP (6,step_image) = mean2(CP21_f_temp);
        CP22_f_temp = CP22_f (step_p_min:step_p_max);
        CP (4,step_image) = mean2(CP22_f_temp);
        CP23_f_temp = CP23_f (step_p_min:step_p_max);
        CP (5,step_image) = mean2(CP23_f_temp);
        %CP24_f_temp = CP24_f (step_p_min:step_p_max);
        %CP (9,step_image) = mean2(CP24_f_temp);
        CP25_f_temp = CP25_f (step_p_min:step_p_max);
        CP (6,step_image) = mean2(CP25_f_temp);

        %3rd row
        CP31_f_temp = CP31_f (step_p_min:step_p_max);
        CP (7,step_image) = mean2(CP31_f_temp);
        CP32_f_temp = CP32_f (step_p_min:step_p_max);
        CP (8,step_image) = mean2(CP32_f_temp);
        CP33_f_temp = CP33_f (step_p_min:step_p_max);
        CP (9,step_image) = mean2(CP33_f_temp);
        CP34_f_temp = CP34_f (step_p_min:step_p_max);
        CP (10,step_image) = mean2(CP34_f_temp);
        %CP35_f_temp = CP35_f (step_p_min:step_p_max);
        %CP (15,step_image) = mean2(CP35_f_temp);

        % next point for time-data
        step_image = step_image + 1;
        if (step_image > size(t_r2,2))
            break;
        end
    end
end

% Load background image
img = imread('run2background.png');
min_x = 0;
max_x = 600;
min_y = 0;
max_y = 200;

fig1 = figure('Position',[500 100 1200 350],'Color',[1 1 1]); %500 left coord. 100 bottom↙
coord. 1200 width 400 height
winsize = get(fig1,'Position'); %set position, dimension of figure for future reference.
winsize(1:2) = [0 0]; %adjust windows size for recording
numframes=size(CP,2); %determines number of frames
set(fig1,'nextplot','replacechildren');
set(gcf,'Renderer','zbuffer');

image = 'Run2/Run2_%03d'; %set name for saving .png-files
        pvinj = 'Emulsion inj. = %1.4f PV'; %set name for legends
```

```
writerObj = VideoWriter('run2.avi'); %prepares .avi-file for saving frames
open(writerObj); %opens .avi-file for saving

for i=1:numframes
    % Get current data for plot
    z_p=CP(:,i);
    Data = scatteredInterpolant(x_crd,y_crd,z_p); %converts data for sutable format
    x_n = 50:10:550;
    y_n = 40:10:160;
    [x_nn,y_nn] = meshgrid(x_n,y_n);
    z_nn = Data (x_nn,y_nn);
   % hold off;
   % imagesc([min_x max_x], [min_y max_y], flipud(img));
   % hold on;
    contourf(x_nn,y_nn,z_nn,50); % plot countor plot

    %set(h,'EdgeColor','none');
    % grid on
   % set(gca,'ydir','normal'); % restore normal axis after loading background

        % Settings for labes, colorbar, axis, title and legend:
        axis([0,600,0,200]) % set axis
        caxis([0, 65])
        set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create xlabel
        xlabel('x coodrinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create ylabel
        ylabel('y coordinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create colorbar
        colorbar('FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create title
        title('Pressure map','FontWeight','bold','FontSize',14,'FontName','Arial');
        legend(sprintf(pvinj ,PV_2(i)),'Location','SouthEast');
        legend('boxoff');
    frame = getframe(fig1,winsize); %get window size
    writeVideo(writerObj,frame);    %write figure in avi-file
    %saveas(fig1,sprintf(image,i),'fig') %save a matlab-fig file
    set(gcf,'PaperPositionMode','auto') % freeze configuration for image, to save image↙
as it is shown on the screen
    print(fig1,'-dpng',sprintf(image,i))
end
close(writerObj);

figure
plot(Timeinmin,CP15_f)


% References:
```

```
% 1. Videowriter
% http://www.mathworks.com/help/matlab/ref/videowriter-class.html
% 2. Creating movies in MATLAB
% http://www.math.canterbury.ac.nz/~c.scarrott/MATLAB_Movies/movies.html
% 3. Background image
% http://www.peteryu.ca/tutorials/matlab/plot_over_image_background
% 4. Save image as png-file
% http://www.mathworks.com/help/matlab/ref/print.html?searchHighlight=print
```

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    Run3_emulsion_data.log
%
%
%
clear
clc


%% Initialize variables.
filename = 'Run3_emulsion_data.log';
delimiter = '\t';

%% Read columns of data as strings:
% For more information, see the TEXTSCAN documentation.
formatSpec = '%*s%*s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%↙
*s%*s%*s%s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%*s%↙
*s%*s%*s%*s%*s%s%*s%*s%*s%*s%*s%*s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,  'ReturnOnError',↙
false);

%% Close the text file.
fclose(fileID);

%% Convert the contents of columns containing numeric strings to numbers.
% Replace non-numeric strings with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = dataArray{col};
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
    % Converts strings in the input cell array to numbers. Replaced non-numeric
    % strings with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1);
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-+]↙
*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
```

```
        try
            result = regexp(rawData{row}, regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if any(numbers==',');
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(thousandsRegExp, ',', 'once'));
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric strings to numbers.
            if ~invalidThousandsSeparator;
                numbers = textscan(strrep(numbers, ',', ''), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch me
        end
    end
end


%% Replace non-numeric cells with NaN
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric cells
raw(R) = {NaN}; % Replace non-numeric cells

%% Allocate imported array to column variable names
Timeinmin = cell2mat(raw(:, 1));
PoreVolumePV = cell2mat(raw(:, 2));
Pressureinlet = cell2mat(raw(:, 3));
EMULSIONIN = cell2mat(raw(:, 4));
CP11 = cell2mat(raw(:, 5));
CP12 = cell2mat(raw(:, 6));
CP13 = cell2mat(raw(:, 7));
CP14 = cell2mat(raw(:, 8));
CP15 = cell2mat(raw(:, 9));
CP21 = cell2mat(raw(:, 10));
CP22 = cell2mat(raw(:, 11));
CP23 = cell2mat(raw(:, 12));
CP24 = cell2mat(raw(:, 13));
CP25 = cell2mat(raw(:, 14));
CP31 = cell2mat(raw(:, 15));
CP32 = cell2mat(raw(:, 16));
CP33 = cell2mat(raw(:, 17));
CP34 = cell2mat(raw(:, 18));
CP35 = cell2mat(raw(:, 19));
```

```
EMULSIONIN (1) = 0;
PoreVolumePV(1) = PoreVolumePV(2);
Pressureinlet(1) = Pressureinlet(2);
Timeinmin (1) = 0;

%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans raw col numericData rawData↙
row regexstr result numbers invalidThousandsSeparator thousandsRegExp me R;

CP11_f = zeros (size(CP11,1),1);
CP12_f = zeros (size(CP12,1),1);
CP13_f = zeros (size(CP13,1),1);
CP14_f = zeros (size(CP14,1),1);
CP15_f = zeros (size(CP15,1),1);
CP21_f = zeros (size(CP21,1),1);
CP22_f = zeros (size(CP22,1),1);
CP23_f = zeros (size(CP23,1),1);
CP24_f = zeros (size(CP24,1),1);
CP25_f = zeros (size(CP25,1),1);
CP31_f = zeros (size(CP31,1),1);
CP32_f = zeros (size(CP32,1),1);
CP33_f = zeros (size(CP33,1),1);
CP34_f = zeros (size(CP34,1),1);
CP35_f = zeros (size(CP35,1),1);

CP11_f(:) = - CP11(:);
CP12_f(:) = - CP12(:);
CP13_f(:) = - CP13(:);
CP14_f(:) = - CP14(:);
CP15_f(:) = - CP15(:);
CP21_f(:) = - CP21(:);
CP22_f(:) = - CP22(:);
CP23_f(:) = - CP23(:);
CP24_f(:) = - CP24(:);
CP25_f(:) = - CP25(:);
CP31_f(:) = - CP31(:);
CP32_f(:) = - CP32(:);
CP33_f(:) = - CP33(:);
CP34_f(:) = - CP34(:);
CP35_f(:) = - CP35(:);

clearvars CP11 CP12 CP13 CP14 CP15 CP21 CP22 CP23 CP24 CP25 CP31 CP32 CP33 CP34 CP35;

% load data from image analysis
load('PVandTime_from_images.mat');

clearvars PV_1 PV_2 t_r1 t_r2;

% temp time for double check
temp_time = zeros (size(t_r3,2),1);
```

```
%% set coordinates for surface plot
x_crd = zeros (10,1); %set to 10, since 5 pressure ports are out
x_crd(1) = 50;
x_crd(2) = 175;
x_crd(3) = 300;
%x_crd(3) = 425;
x_crd(4) = 550;

x_crd(5) = 50;
%x_crd(5) = 175;
%x_crd(8) = 300;
x_crd(6) = 425;
%x_crd(6) = 550;

%x_crd(7) = 50;
x_crd(7) = 175;
x_crd(8) = 300;
x_crd(9) = 425;
x_crd(10) = 550;

y_crd = zeros (10,1);
y_crd(1) = 160;
y_crd(2) = 160;
y_crd(3) = 160;
%y_crd(3) = 160;
y_crd(4) = 160;

y_crd(5) = 100;
%y_crd(5) = 100;
%y_crd(8) = 100;
y_crd(6) = 100;
%y_crd(6) = 100;

%y_crd(7) = 40;
y_crd(7) = 40;
y_crd(8) = 40;
y_crd(9) = 40;
y_crd(10) = 40;

CP = zeros (10,size(t_r3,2));
%%
step_image = 2;
for step = 1 : size (Timeinmin,1)
    if (Timeinmin(step) >=t_r3(step_image))
        % save current time for further reference
        temp_time (step_image) = Timeinmin(step,1);

        %Set new step for pressure data
        step_p_min = step - 10;
        step_p_max = step + 10;
```

```
        %Trim new data for plotting +/- 10 steps from given point
        %1st row
        CP11_f_temp = CP11_f (step_p_min:step_p_max);
        CP (1,step_image) = mean2(CP11_f_temp);
        CP12_f_temp = CP12_f (step_p_min:step_p_max);
        CP (2,step_image) = mean2(CP12_f_temp);
        CP13_f_temp = CP13_f (step_p_min:step_p_max);
        CP (3,step_image) = mean2(CP13_f_temp);
        %CP14_f_temp = CP14_f (step_p_min:step_p_max);
        %CP (3,step_image) = mean2(CP14_f_temp);
        CP15_f_temp = CP15_f (step_p_min:step_p_max);
        CP (4,step_image) = mean2(CP15_f_temp);

        %2nd row
        CP21_f_temp = CP21_f (step_p_min:step_p_max);
        CP (5,step_image) = mean2(CP21_f_temp);
        %CP22_f_temp = CP22_f (step_p_min:step_p_max);
        %CP (5,step_image) = mean2(CP22_f_temp);
        %CP23_f_temp = CP23_f (step_p_min:step_p_max);
        %CP (8,step_image) = mean2(CP23_f_temp);
        CP24_f_temp = CP24_f (step_p_min:step_p_max);
        CP (6,step_image) = mean2(CP24_f_temp);
        %CP25_f_temp = CP25_f (step_p_min:step_p_max);
        %CP (6,step_image) = mean2(CP25_f_temp);

        %3rd row
        %CP31_f_temp = CP31_f (step_p_min:step_p_max);
        %CP (7,step_image) = mean2(CP31_f_temp);
        CP32_f_temp = CP32_f (step_p_min:step_p_max);
        CP (7,step_image) = mean2(CP32_f_temp);
        CP33_f_temp = CP33_f (step_p_min:step_p_max);
        CP (8,step_image) = mean2(CP33_f_temp);
        CP34_f_temp = CP34_f (step_p_min:step_p_max);
        CP (9,step_image) = mean2(CP34_f_temp);
        CP35_f_temp = CP35_f (step_p_min:step_p_max);
        CP (10,step_image) = mean2(CP35_f_temp);

        % next point for time-data
        step_image = step_image + 1;
        if (step_image > size(t_r3,2))
            break;
        end
    end
end

% Load background image
img = imread('run3background.png');
min_x = 0;
max_x = 600;
min_y = 0;
max_y = 200;
```

```
fig1 = figure('Position',[500 100 1200 350],'Color',[1 1 1]); %500 left coord. 100 bottom coord. 1200 width 400 height
winsize = get(fig1,'Position'); %set position, dimension of figure for future reference.
winsize(1:2) = [0 0]; %adjust windows size for recording
numframes=size(CP,2); %determines number of frames
set(fig1,'nextplot','replacechildren');
set(gcf,'Renderer','zbuffer');

image = 'Run3/Run3_%03d'; %set name for saving .png-files
        pvinj = 'Emulsion inj. = %1.4f PV'; %set name for legends

writerObj = VideoWriter('run3.avi'); %prepares .avi-file for saving frames
open(writerObj); %opens .avi-file for saving

for i=1:numframes
    % Get current data for plot
    z_p=CP(:,i);
    Data = scatteredInterpolant(x_crd,y_crd,z_p); %converts data for sutable format
    x_n = 50:10:550;
    y_n = 40:10:160;
    [x_nn,y_nn] = meshgrid(x_n,y_n);
    z_nn = Data (x_nn,y_nn);
    hold off;
    imagesc([min_x max_x], [min_y max_y], flipud(img));
    hold on;
    contourf(x_nn,y_nn,z_nn,50); % plot countor plot

    %set(h,'EdgeColor','none');
    % grid on
    set(gca,'ydir','normal'); % restore normal axis after loading background

        % Settings for labes, colorbar, axis, title and legend:
        axis([0,600,0,200]) % set axis
        caxis([0, 115])
        set(gca,'color','w','FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create xlabel
        xlabel('x coodrinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create ylabel
        ylabel('y coordinate, mm','FontWeight','bold','FontSize',12,...
        'FontName','Arial');
        % Create colorbar
        colorbar('FontWeight','bold','FontSize',14,'FontName','Arial');
        % Create title
        title('Pressure map','FontWeight','bold','FontSize',14,'FontName','Arial');
        legend(sprintf(pvinj ,PV_3(i)),'Location','SouthEast');
        legend('boxoff');
    frame = getframe(fig1,winsize); %get window size
    writeVideo(writerObj,frame);    %write figure in avi-file
    %saveas(fig1,sprintf(image,i),'fig') %save a matlab-fig file
```

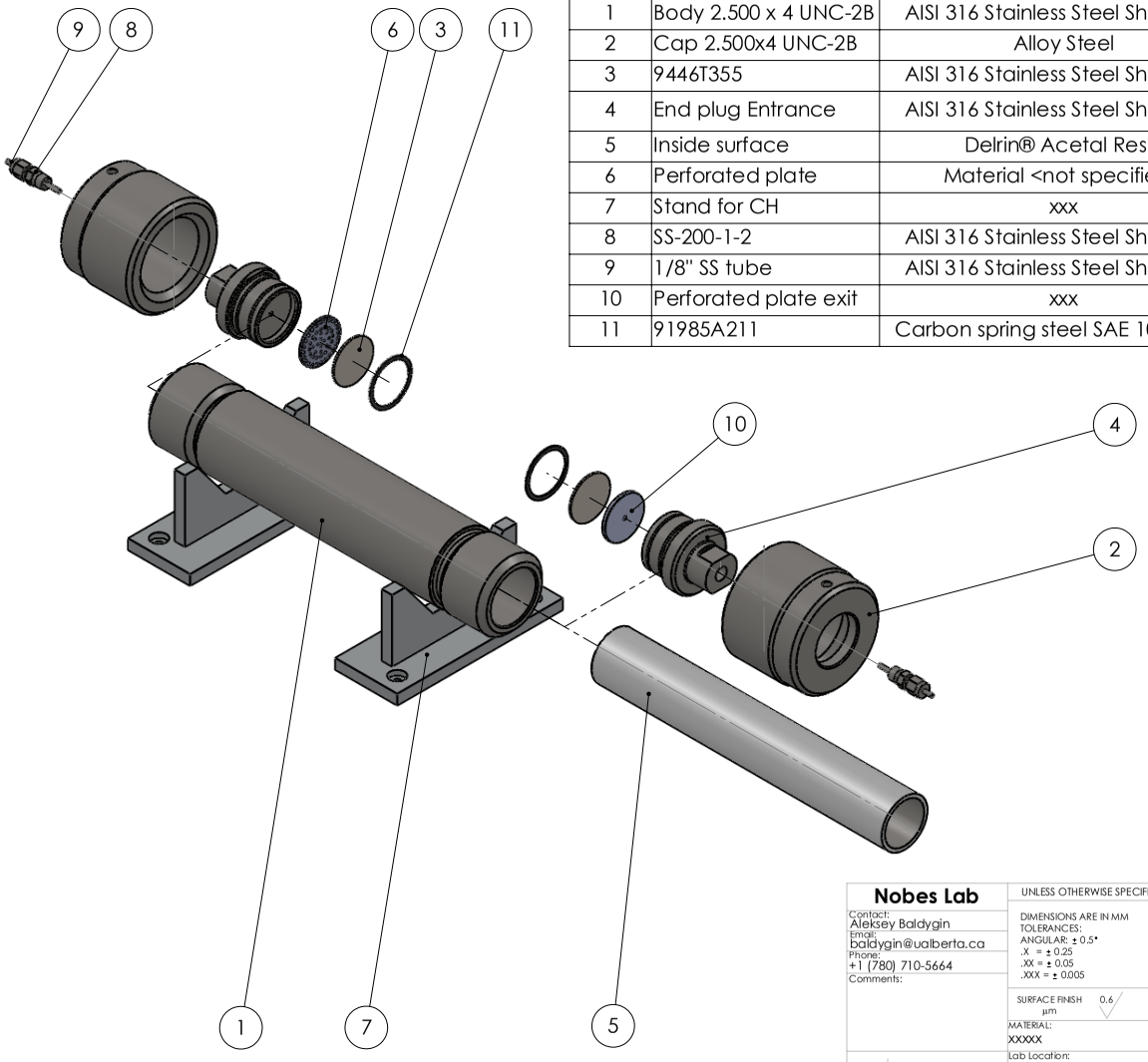
```
    set(gcf,'PaperPositionMode','auto') % freeze configuration for image, to save image↙
as it is shown on the screen
    print(fig1,'-dpng',sprintf(image,i))
end
close(writerObj);


% References:
% 1. Videowriter
% http://www.mathworks.com/help/matlab/ref/videowriter-class.html
% 2. Creating movies in MATLAB
% http://www.math.canterbury.ac.nz/~c.scarrott/MATLAB_Movies/movies.html
% 3. Background image
% http://www.peteryu.ca/tutorials/matlab/plot_over_image_background
% 4. Save image as png-file
% http://www.mathworks.com/help/matlab/ref/print.html?searchHighlight=print
```

A-6 Set of drawings

The following set of drawings was prepared using computer-aided design software (CAD) (SolidWorks 2011-2014, Dassault Systèmes SolidWorks Corp.).

ITEM NO.	SW-Title(Title)	Unit	Supplier	QTY.
1	Body 2.500 x 4 UNC-2B	AISI 316 Stainless Steel Sheet (SS)	MecE machine shop	1
2	Cap 2.500x4 UNC-2B	Alloy Steel	MecE machine shop	2
3	9446T355	AISI 316 Stainless Steel Sheet (SS)	Applied POROUS Technologies Inc.	2
4	End plug Entrance	AISI 316 Stainless Steel Sheet (SS)	MecE machine shop	2
5	Inside surface	Delrin® Acetal Resin	MecE machine shop	1
6	Perforated plate	Material <not specified>	Prototype printer	1
7	Stand for CH	xxx	MecE machine shop	2
8	SS-200-1-2	AISI 316 Stainless Steel Sheet (SS)	Swagelock	2
9	1/8" SS tube	AISI 316 Stainless Steel Sheet (SS)	Swagelock	2
10	Perforated plate exit	xxx	Prototype printer	1
11	91985A211	Carbon spring steel SAE 1070-1090	McMaster-Carr	2



Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: +1 (780) 710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH 0.6 µm
MATERIAL: XXXXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	**xxxxx**
MecE Job #	**11022**
File Name:	2.500 x 4 UNC-2A 2B Core holder

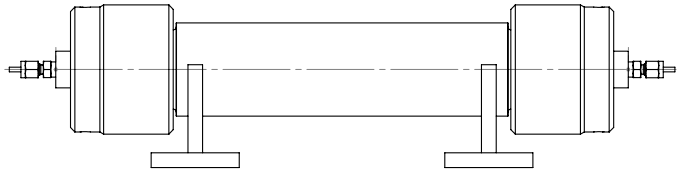
baldygin
9/19/2014 4:18:25 PM
Friday, September 19, 2014

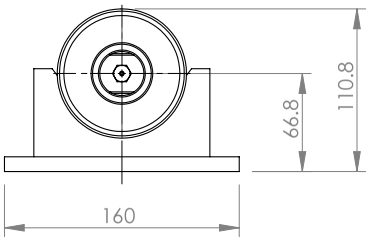
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

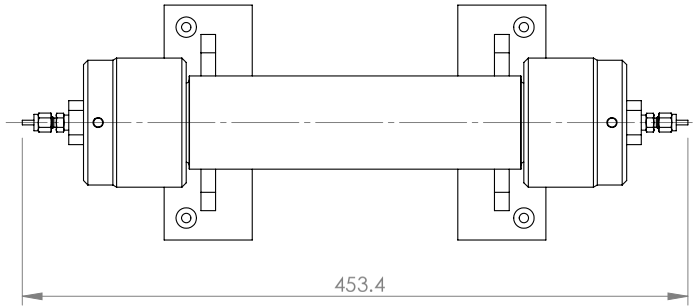
TITLE:
Core holder 2 1/2-4 UNC - 2A/2B

SIZE: B | Project: Quadrise project | REV: XXX
SCALE: 1:3 | Mass: 9759.97 | SHEET 1 OF 9

ITEM NO.	SW-Title(Title)	Supplier	QTY.
1	91985A211	McMaster-Carr	2
2	9446T355	Applied POROUS Technologies Inc.	2
3	End plug Entrance	MecE machine shop	2
4	Inside surface	MecE machine shop	1
5	Perforated plate	Prototype printer	1
6	Stand for CH	MecE machine shop	2
7	SS-200-1-2	Swagelock	2
8	1/8" SS tube	Swagelock	2
9	Perforated plate exit	Prototype printer	1
10	Body 2.500 x 4 UNC-2B	MecE machine shop	1
11	Cap 2.500x4 UNC-2B	MecE machine shop	2







Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: +1 (780) 710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH 0.6 µm
MATERIAL: XXXXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

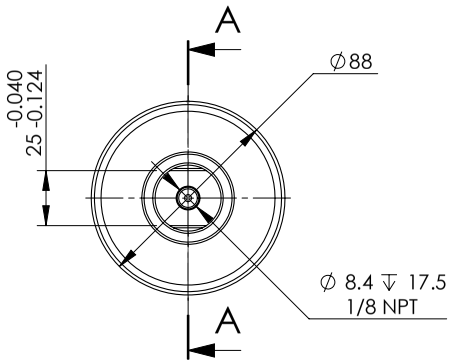
	NAME
DRAWN	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	**xxxxx**
MecE Job #	**11022**
File Name:	2.500 x 4 UNC-2A 2B Core holder

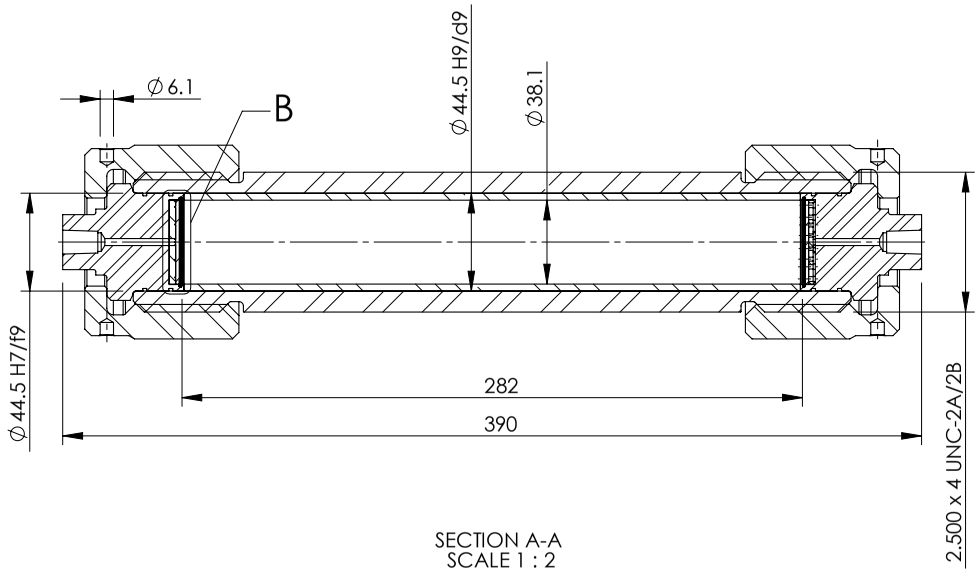
baldygin
9/19/2014 4:18:25 PM
Friday, September 19, 2014

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

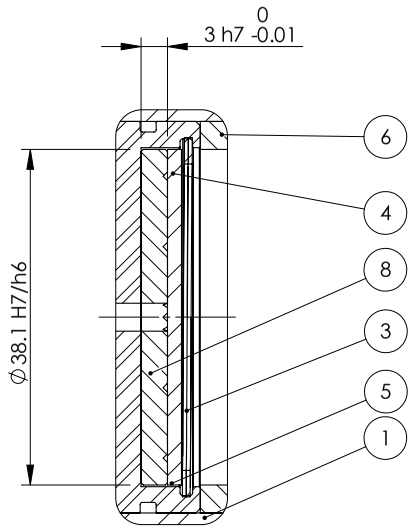
TITLE:
Core holder 2 1/2-4 UNC - 2A/2B

SIZE: B | Project: Quadrise project | REV: XXX
SCALE: 1:3 | Mass: 9759.97 | SHEET 2 OF 9





SECTION A-A
SCALE 1 : 2



DETAIL B
SCALE 2 : 1

ITEM NO.	SW-Title(Title)	Supplier	QTY.
1	Body 2.500 x 4 UNC-2B	MecE machine shop	1
2	Cap 2.500x4 UNC-2B	MecE machine shop	2
3	91985A211	McMaster-Carr	2
4	9446T355	Applied POROUS Technologies Inc.	2
5	End plug Entrance	MecE machine shop	2
6	Inside surface	MecE machine shop	1
7	Perforated plate	Prototype printer	1
8	Perforated plate exit	Prototype printer	1

Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: +1 (780) 710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH 0.6 µm
MATERIAL: XXXXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	xxxxx
MecE Job #	xxx
File Name:	2.500 x 4 UNC-2A 2B Core holder without stand

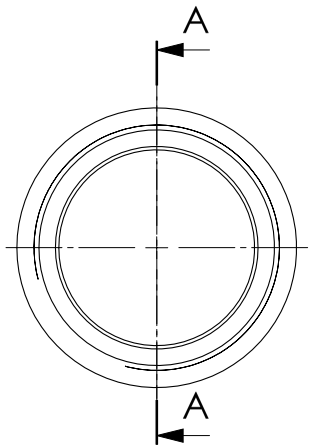
baldygin
9/19/2014 4:18:25 PM
Friday, September 19, 2014

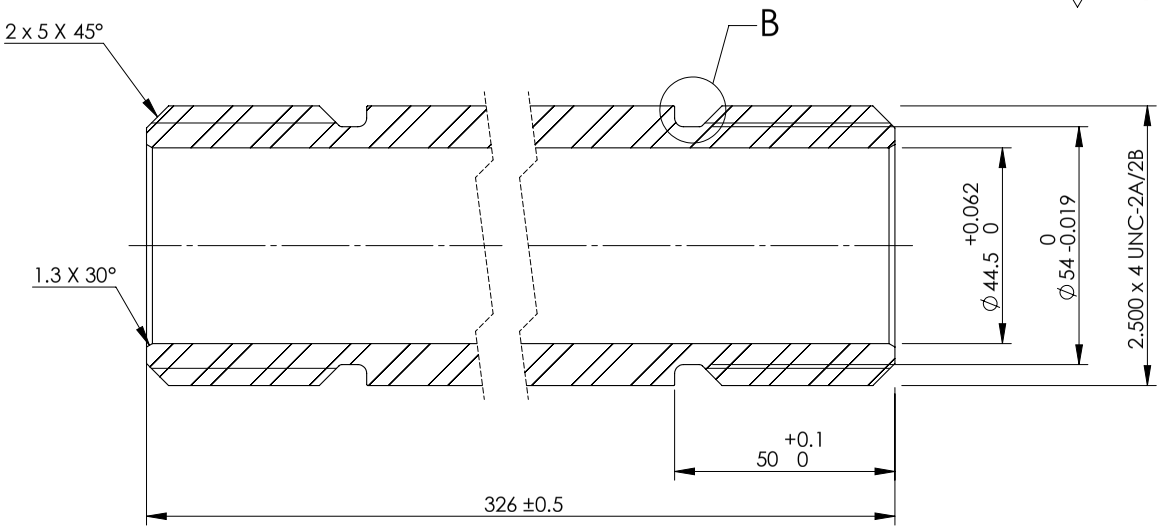
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Core holder

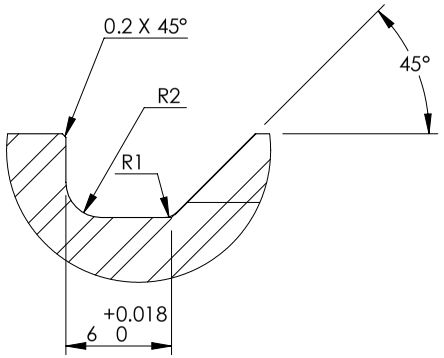
SIZE: B | Project: Quadrise project | REV: xxx

SCALE: 1:10 | Mass: 8930.72 | SHEET 3 OF 9





SECTION A-A



DETAIL B
SCALE 4 : 1

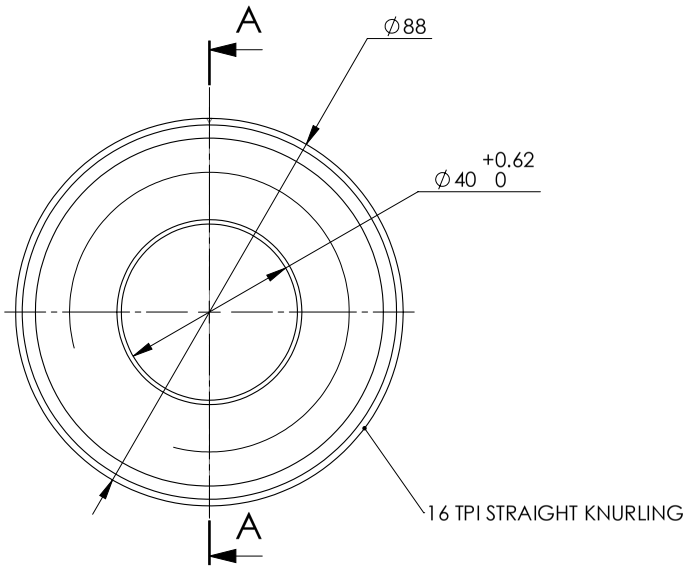
Ra 3.2

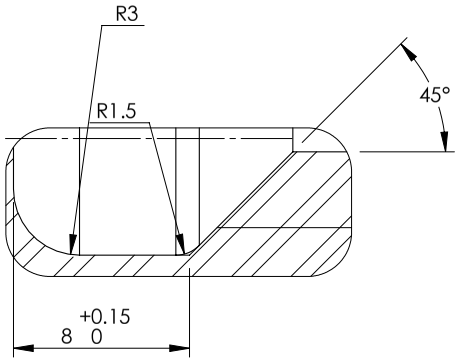
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005	DRAWN	Aleksey Baldygin
Email: baldygin@ualberta.ca		CHECKED	Dr. David Nobes
Phone: (780) 710-5664		ENG APPR.	xxx
Comments:		**Speed Code**	**xxx**
	SURFACE FINISH 0.6 µm	**MecE Job #**	**11022**
	MATERIAL: AISI 316 Stainless Steel Sheet (SS)	File Name:	2.500 x 4 UNC-2A 2B Body
	Lab Location: NINT 6-93	baldygin 9/19/2014 4:18:25 PM Friday, September 19, 2014	
	DO NOT SCALE DRAWING		

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

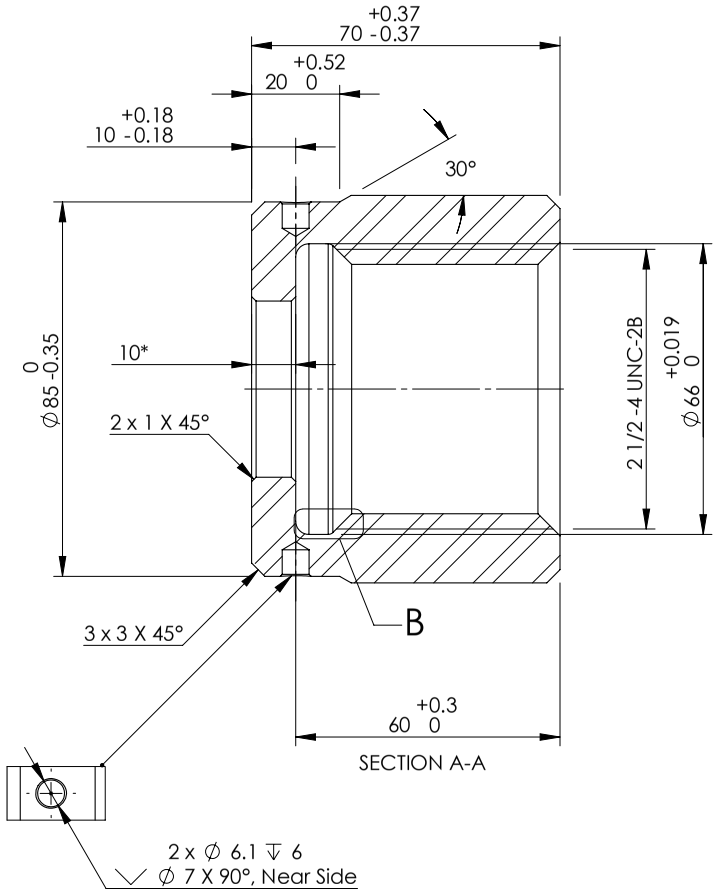
TITLE: Body 2.500 x 4 UNC-2B

SIZE	Project:	REV
B	Quadrise Project	xxx
SCALE: 1:1	Mass: 4057.27	SHEET 4 OF 9





DETAIL B
SCALE 5 : 1



SECTION A-A

Ra 3.2

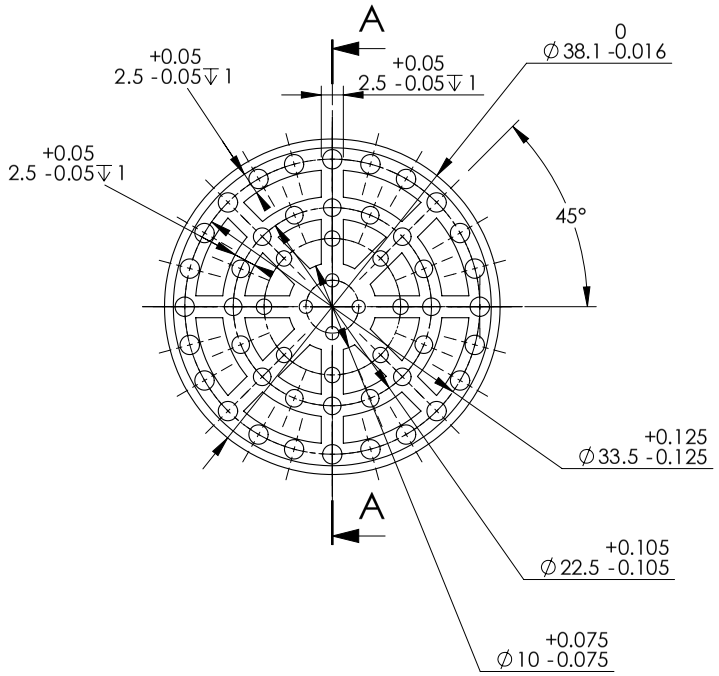
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN	Aleksey Baldygin
Email: baldygin@ualberta.ca		CHECKED	Dr. David Nobes
Phone: (780)710-5664		ENG APPR.	xxx
Comments: Different configuration with 65x4 thread.		**Speed Code**	**xxx**
	INTERPRET GEOMETRIC TOLERANCING PER:	**MecE Job #**	**11022**
	MATERIAL: Alloy Steel	File Name:	2.500 x 4 UNC-2A 2B CAP
	Lab Location: NINT 6-93	baldygin 9/19/2014 4:18:25 PM Friday, September 19, 2014	
	DO NOT SCALE DRAWING		

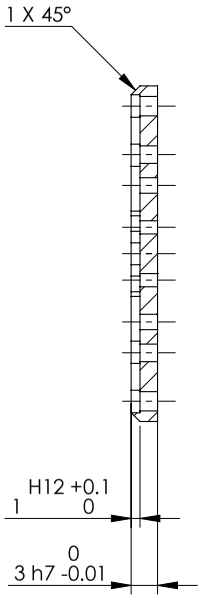
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Cap 2.500x4 UNC-2B

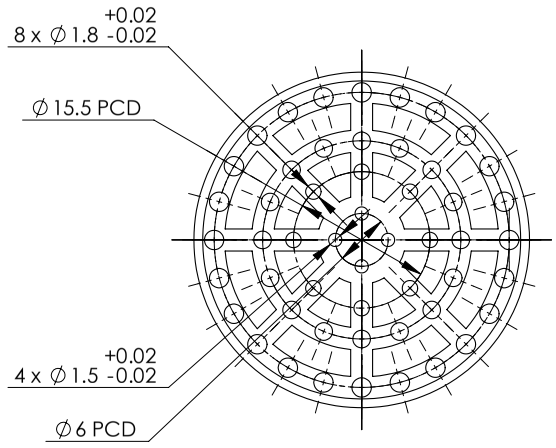
SIZE	Project:	REV
B	Quadrise Project	xxx

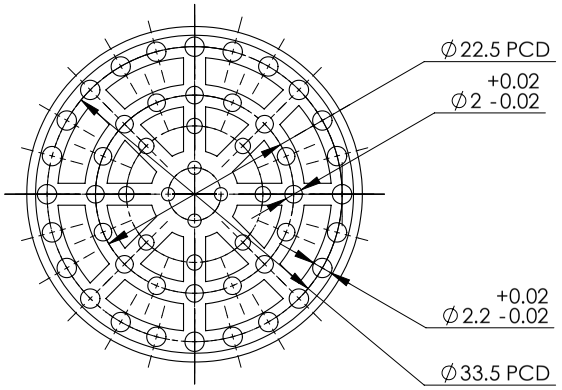
SCALE: 1:1 | Mass: 1842.23 | SHEET 5 OF 9





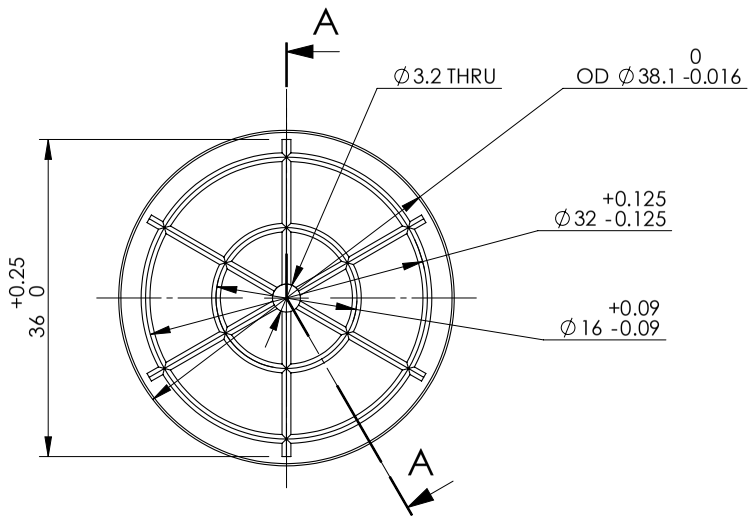
SECTION A-A

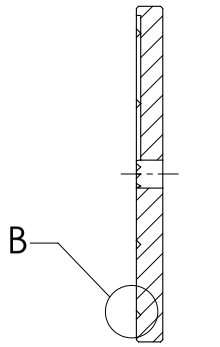




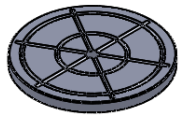
This part will be made on 3D Printing machine

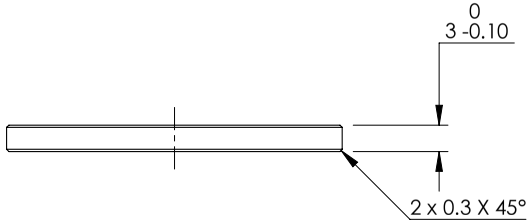
Nobes Lab Contact: Aleksey Baldygin Email: baldygin@ualberta.ca Phone: +1 (780) 710-5664 Comments:	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005 SURFACE FINISH 0.6 µm MATERIAL: Material <not specified> Lab Location: NINT 6-93 DO NOT SCALE DRAWING	NAME DRAWN Aleksey Baldygin CHECKED Dr. David Nobes ENG APPR. xxx **Speed Code** **xxx** **MecE Job #** **xxx** File Name: Perforated_plate baldygin 9/19/2014 4:18:25 PM Friday, September 19, 2014	The Department of Mechanical Engineering UNIVERSITY OF ALBERTA TITLE: Perforated plate SIZE B Project: Quadrise project REV xxx SCALE: 2:1 Mass: 2.35 SHEET 6 OF 9

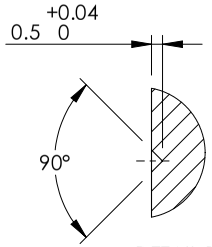




SECTION A-A



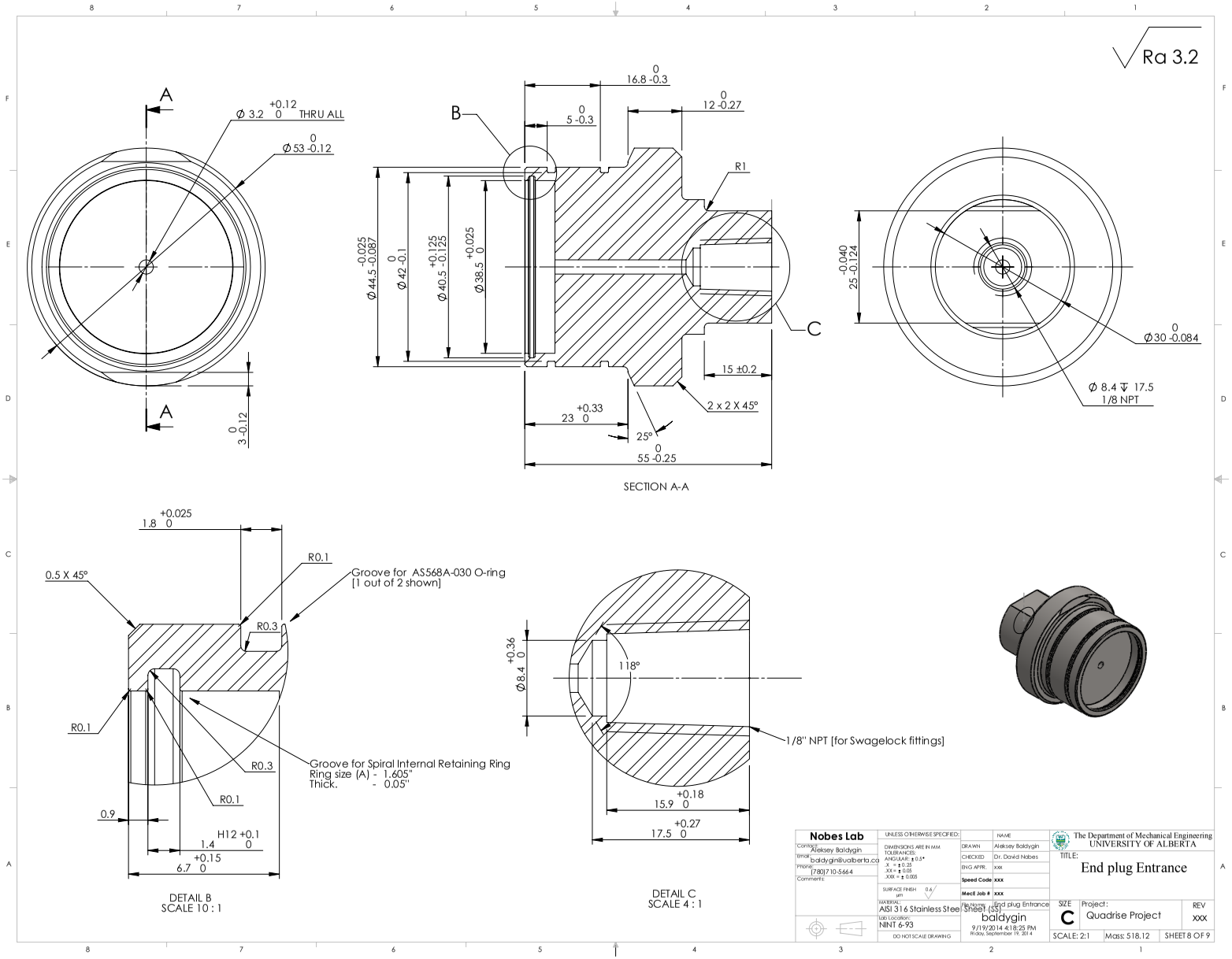




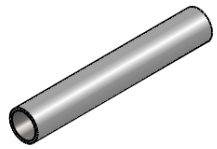
DETAIL B
SCALE 5 : 1

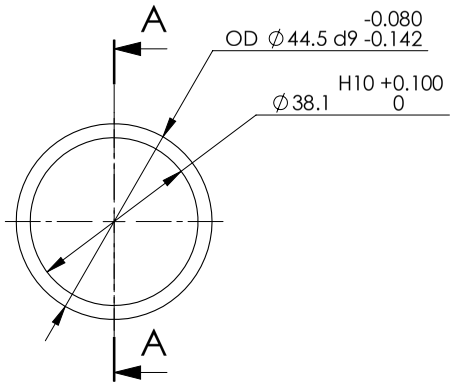
This part will be made on 3D Printing machine

Nobes Lab Contact: Aleksey Baldygin Email: baldygin@ualberta.ca Phone: (780)710-5664 Comments:	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005 SURFACE FINISH 0.6 µm MATERIAL: xxx Lab Location: NINT 6-93 DO NOT SCALE DRAWING	NAME DRAWN Aleksey Baldygin CHECKED Dr. David Nobes ENG APPR. xxx **Speed Code** xxx **MecE Job #** xxx File Name: Perforated_plate_exit baldygin 9/19/2014 4:18:25 PM Friday, September 19, 2014	The Department of Mechanical Engineering UNIVERSITY OF ALBERTA TITLE: Perforated plate exit SIZE B Project: Quadrise project REV xxx SCALE: 2:1 Mass: xxx SHEET 7 OF 9



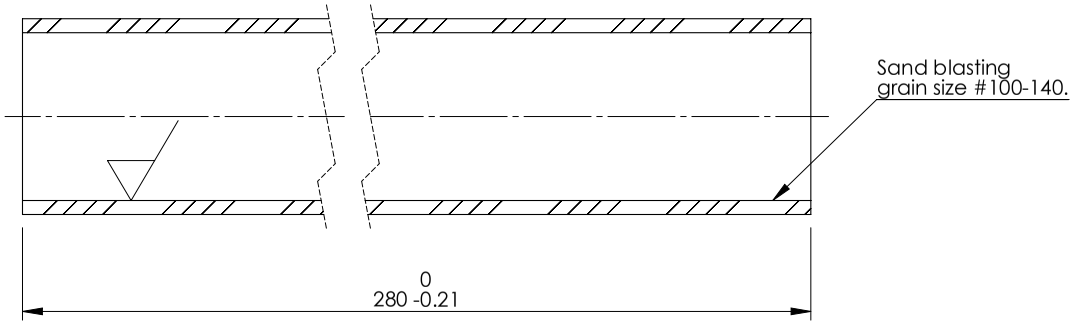
should measure SS filter first, to make a desicion... base on their tolerances





OD ∅44.5 d9 $^{-0.080}_{-0.142}$

∅38.1 H10 $^{+0.100}_{0}$



SECTION A-A

Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005	DRAWN	Aleksey Baldygin
Email: baldygin@ualberta.ca		CHECKED	Dr. David Nobes
Phone: (780)710-5664		ENG APPR.	xxx
Comments:		**Speed Code**	**xxx**
	SURFACE FINISH 0.6 µm	**MecE Job #**	**xxx**
	MATERIAL: Delrin® Acetal Resin	File Name:	Insidesurface
	Lab Location: xxx	baldygin 9/19/2014 4:18:25 PM Friday, September 19, 2014	
	DO NOT SCALE DRAWING		

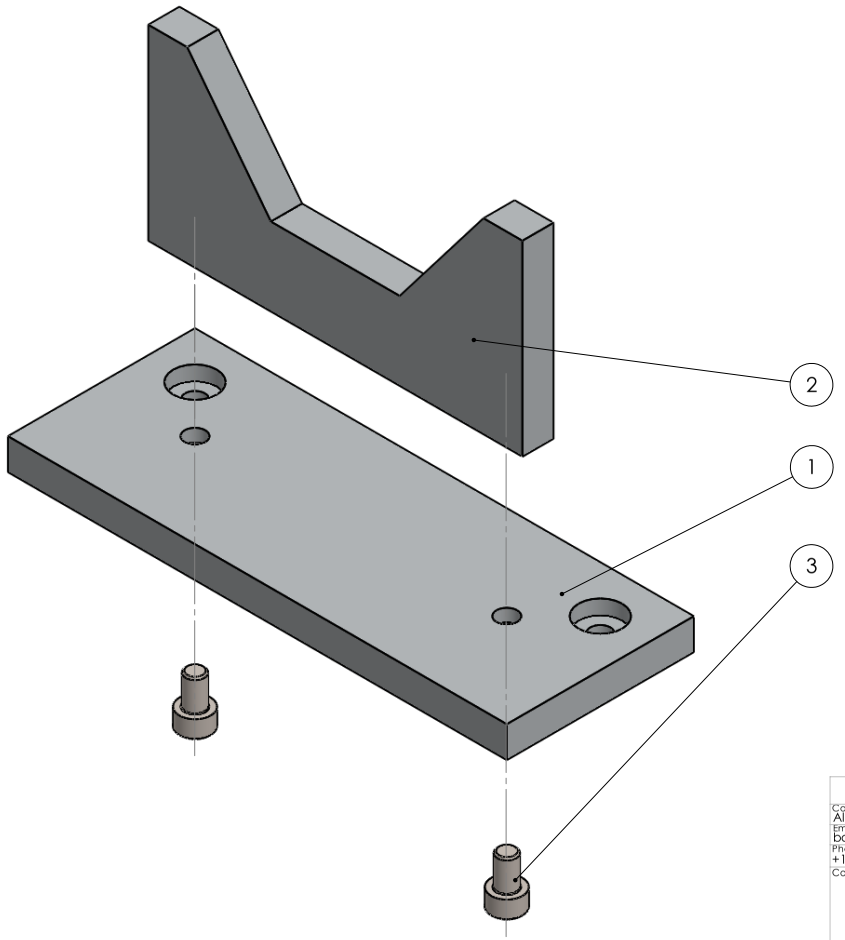
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Inside surface

SIZE	Project:	REV
B	Quadrise Project	xxx

SCALE: 1:1 | Mass: 117.58 | SHEET 9 OF 9

ITEM NO.	PART NUMBER	Material	QTY.
1	Bottom_stand	2014-T4	1
2	vertical_stand	2014-T4	1
3	B18.3.1M - 6 x 1.0 x 10 Hex SHCS -- 10NHX		2



Nobes Lab

Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: +1 (780) 710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005

SURFACE FINISH 0.6 µm

MATERIAL: XXX

Lab Location: NINT 6-93

DO NOT SCALE DRAWING

	NAME
DRAWN	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	**xxx**
MecE Job #	**xxx**
File Name:	Stand_CH

baldygin
12/1/2014 9:35:23 AM
Monday, December 01, 2014

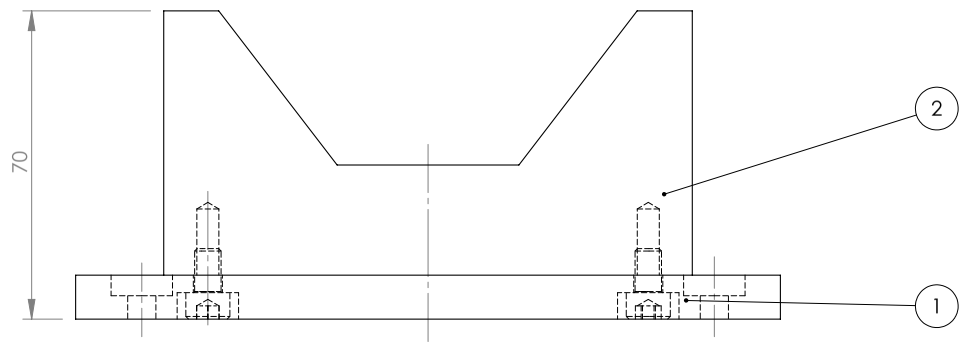
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

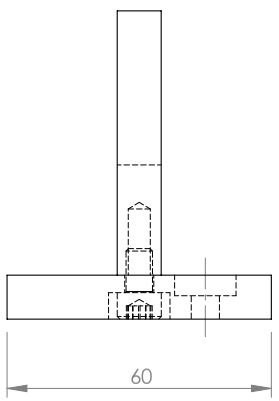
TITLE: Stand for CH

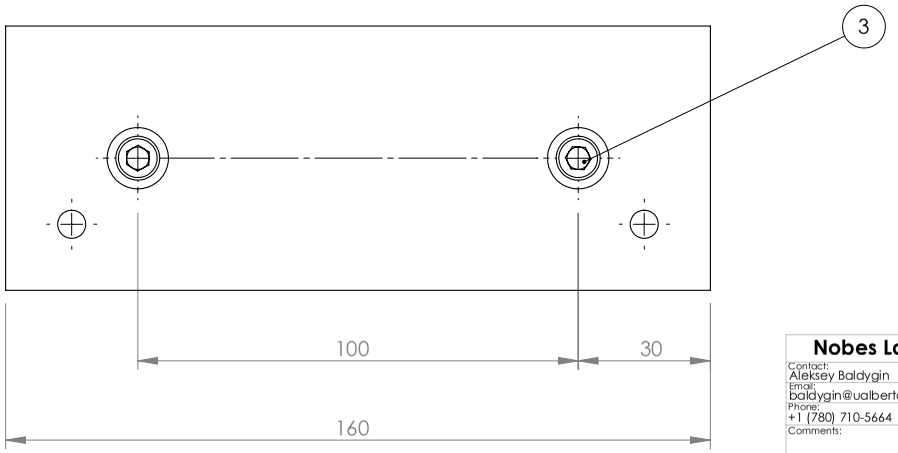
SIZE B | Project: Quadrise Project | REV xxx

SCALE: 1:1 | Mass: | SHEET 1 OF 4

ITEM NO.	PART NUMBER	Material	QTY.
1	Bottom_stand	2014-T4	1
2	vertical_stand	2014-T4	1
3	B18.3.1M - 6 x 1.0 x 10 Hex SHCS -- 10NHX		2







Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: +1 (780) 710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH 0.6 µm
MATERIAL: XXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	xxx
MecE Job #	xxx
File Name:	Stand_CH

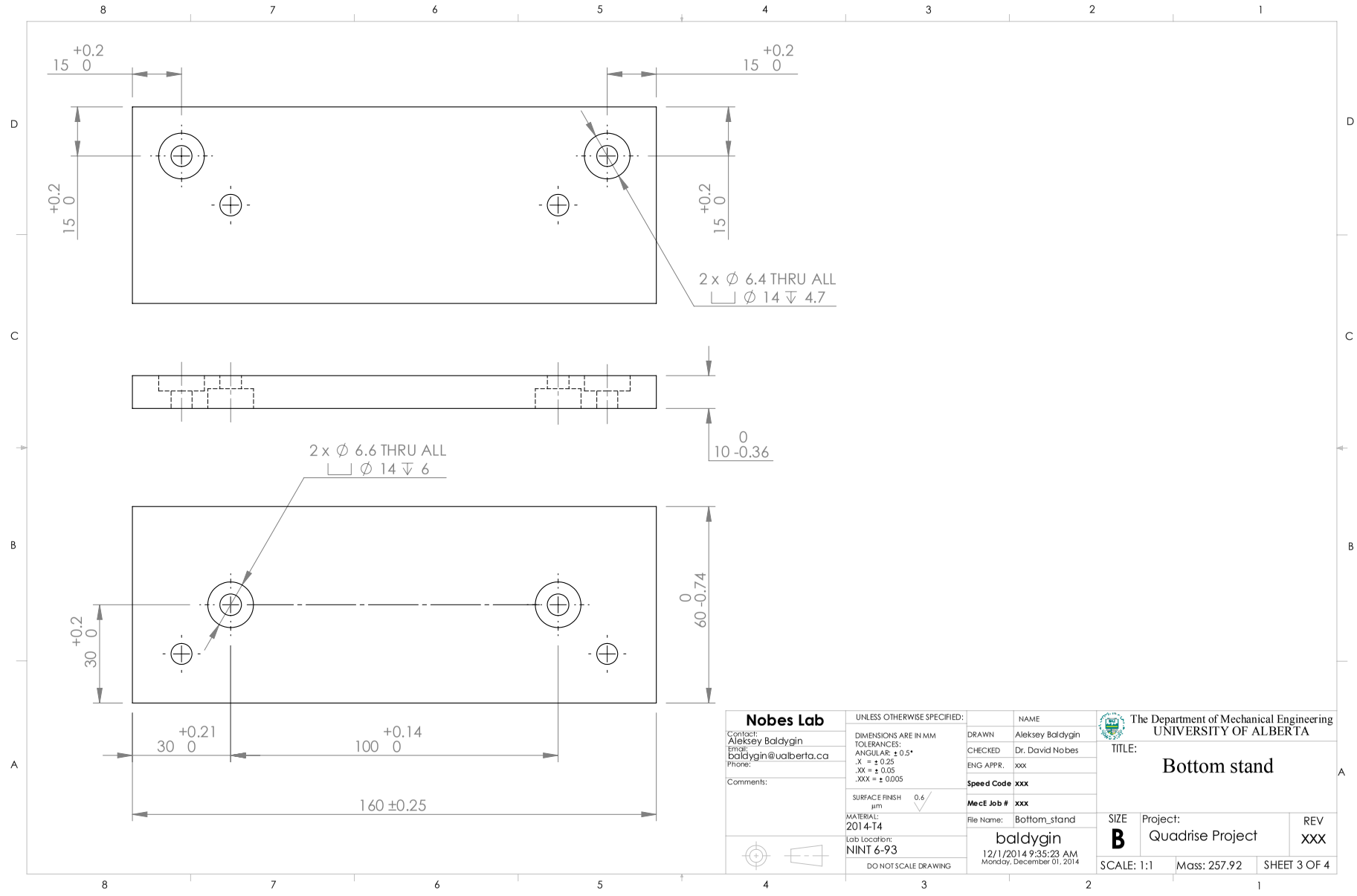
baldygin
12/1/2014 9:35:23 AM
Monday, December 01, 2014

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Stand for CH

SIZE B | Project: Quadrise Project | REV xxx

SCALE: 1:1 | Mass: | SHEET 2 OF 4



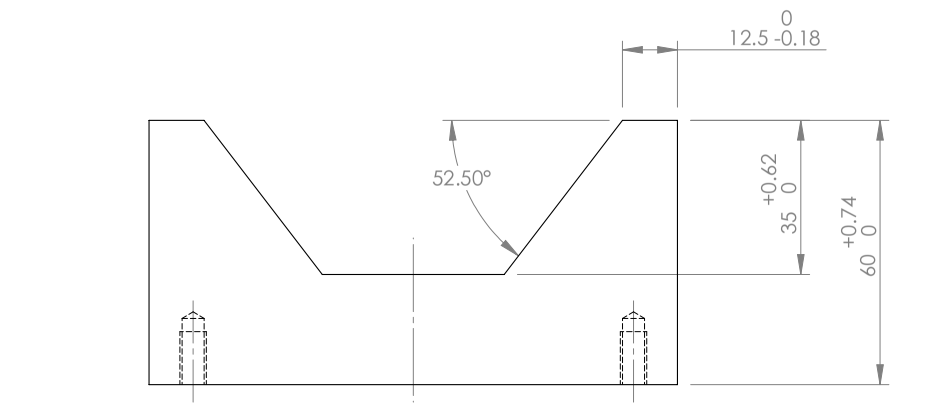
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES:	CHECKED	Dr. David Nobes
Phone:	ANGULAR: ± 0.5°	ENG APPR.	xxx
Comments:	.X = ± 0.25	**Speed Code**	**xxx**
	.XX = ± 0.05	**MecE Job #**	**xxx**
	.XXX = ± 0.005	File Name:	Bottom_stand
	SURFACE FINISH 0.6 µm	baldygin	
	MATERIAL: 2014-T4	12/1/2014 9:35:23 AM	
	Lab Location: NINT 6-93	Monday, December 01, 2014	
	DO NOT SCALE DRAWING		

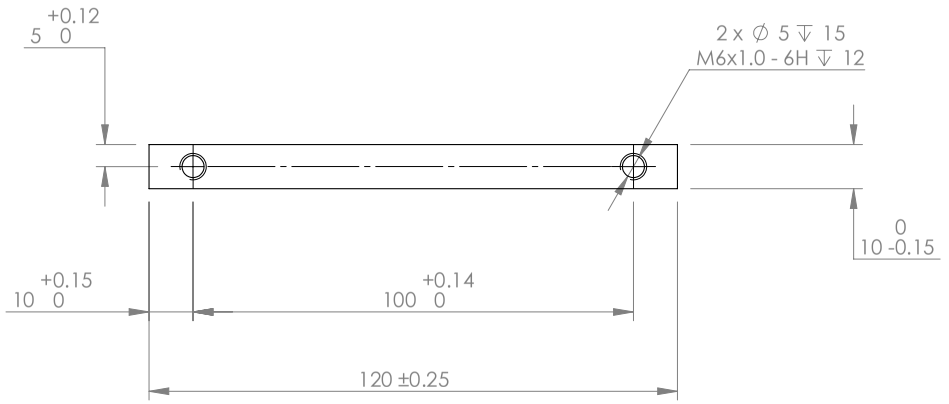
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Bottom stand

SIZE	Project:	REV
B	Quadrise Project	xxx

SCALE: 1:1 | Mass: 257.92 | SHEET 3 OF 4





Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES:	CHECKED	Dr. David Nobes
Phone: (780) 710-5664	ANGULAR: ± 0.5°	ENG APPR.	xxx
Comments:	.X = ± 0.25	**Speed Code**	**xxx**
	.XX = ± 0.05	**MecE Job #**	**xxx**
	.XXX = ± 0.005	File Name:	vertical_stand
	SURFACE FINISH 0.6 µm	baldygin	
	MATERIAL: 2014-T4	12/1/2014 9:35:23 AM	
	Lab Location: NINT 6-93	Monday, December 01, 2014	
	DO NOT SCALE DRAWING		

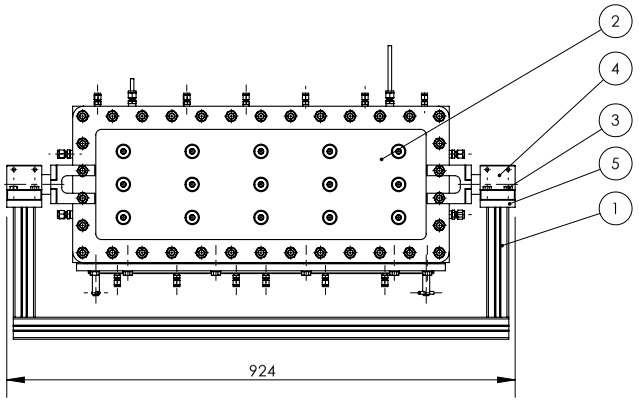
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

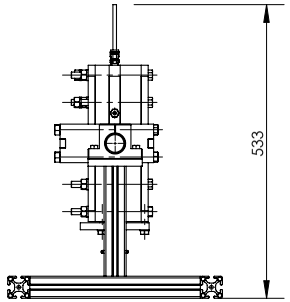
TITLE: Vertical stand

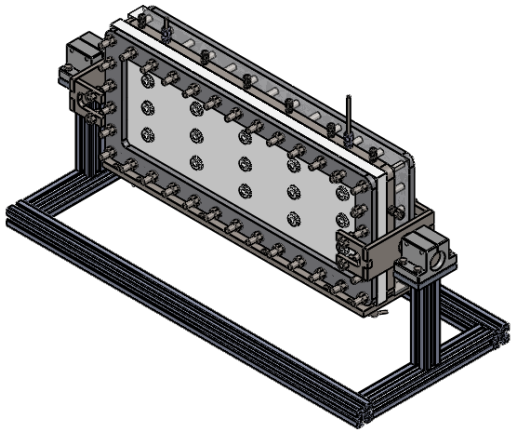
SIZE	Project:	REV
B	Quadrise Project	xxx

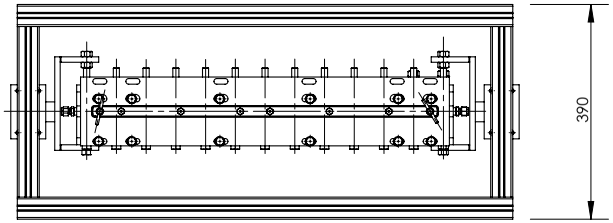
SCALE: 1:1 | Mass: 133.11 | SHEET 4 OF 4

ITEM NO.	SW-Title(Title)	PART NUMBER	Supplier	QTY.
1	Frame from 8020 parts	Frame	8020.net	1
2	2D core holder	2DCH_glass_walls	MECE Machine shop	1
3		HBOLT 0.3125-18x1x0.875-N	McMaster-Carr	8
4	8020-5930	8020-5930	80/20 Inc.	2
5	Plate for 8020 box	plate for 8020box	MECE Machine shop	2









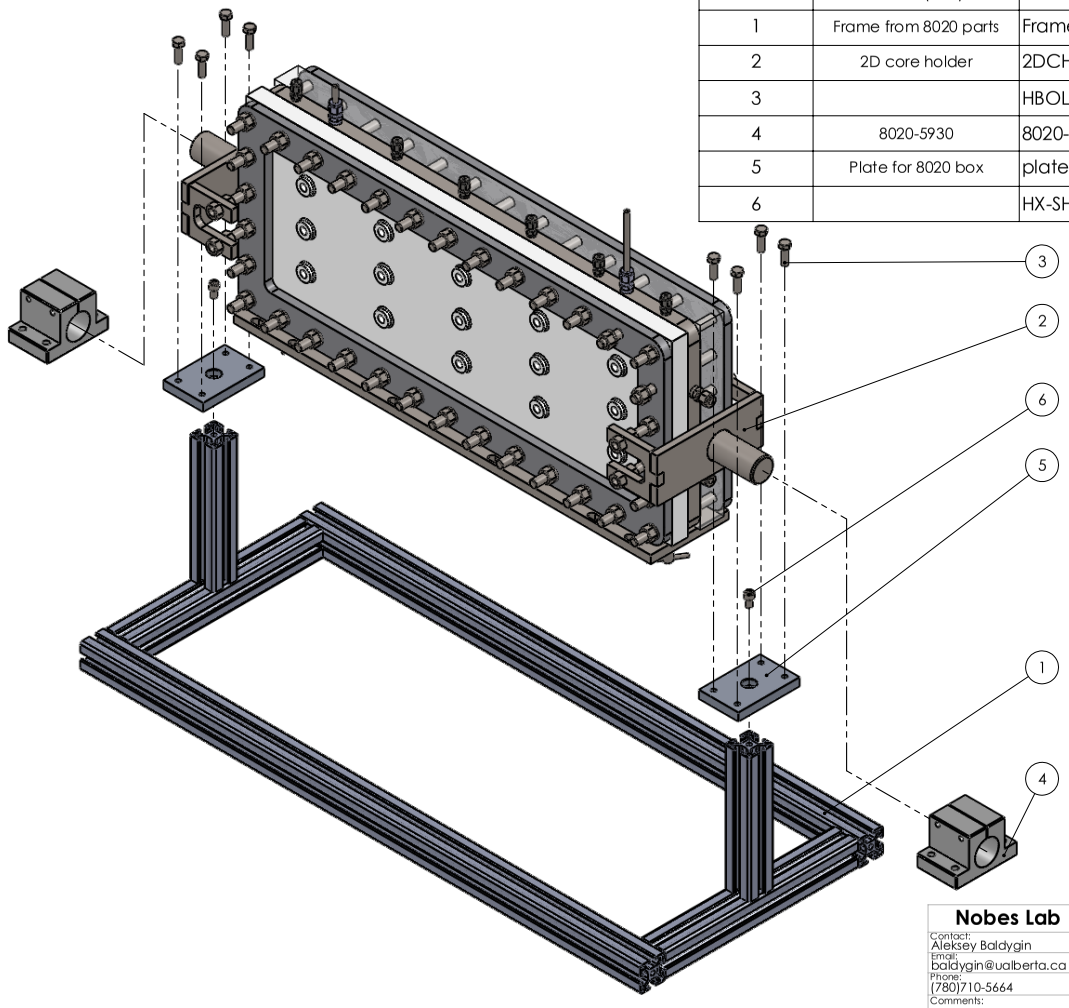
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments:	SURFACE FINISH 0.6 µm	**Speed Code**	**MECE 234**
	MATERIAL: XXXXX	**MecE Job #**	**xxx**
		File Name:	2Dcoreholder_on_frame_autosaved by
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Monday, December 01, 2014 9:47:43 AM Friday, June 21, 2013 10:29:53 AM	

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE:
2D core holder with frame

SIZE	Project:	REV
B	Quadrise	XXX
SCALE: 1:8	Mass: xxxxx	SHEET 1 OF 11

ITEM NO.	SW-Title(Title)	PART NUMBER	Supplier	QTY.
1	Frame from 8020 parts	Frame	8020.net	1
2	2D core holder	2DCH_glass_walls	MECE Machine shop	1
3		HBOLT 0.3125-18x1x0.875-N	McMaster-Carr	8
4	8020-5930	8020-5930	80/20 Inc.	2
5	Plate for 8020 box	plate for 8020box	MECE Machine shop	2
6		HX-SHCS 0.3125-18x0.5x0.5-N	McMaster-Carr	2



Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: (780)710-5664
Comments:

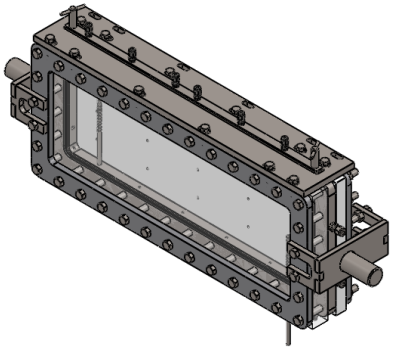
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
SURFACE FINISH 0.6 µm
MATERIAL: XXXXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN BY	Aleksey Baldygin
SMB	Aleksey Baldygin
CHECKED BY	Dr. David Nobes
Speed Code	MECE 234
MecE Job #	xxx
File Name:	2Dcoreholder_on_frame_autosaved by
	baldygin
	Monday, December 01, 2014 9:47:43 AM
	Friday, June 21, 2013 10:29:53 AM

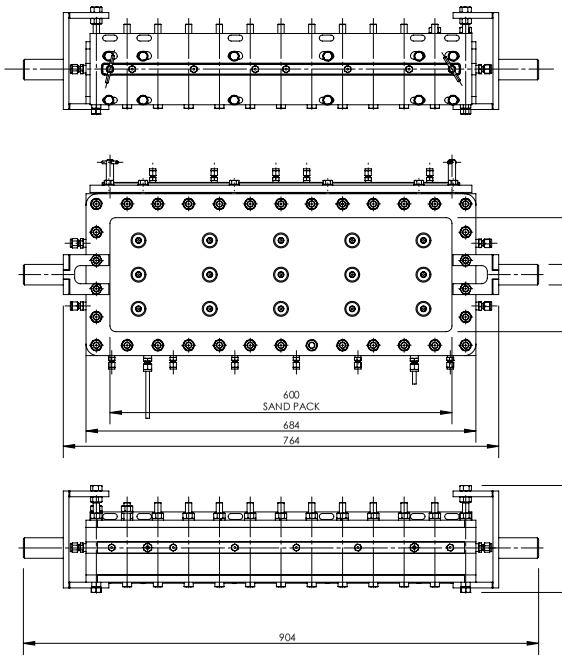
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE:
2D core holder with frame

SIZE: B | Project: Quadrise | REV: XXX
SCALE: 1:5 | Mass: xxxxx | SHEET 2 OF 11

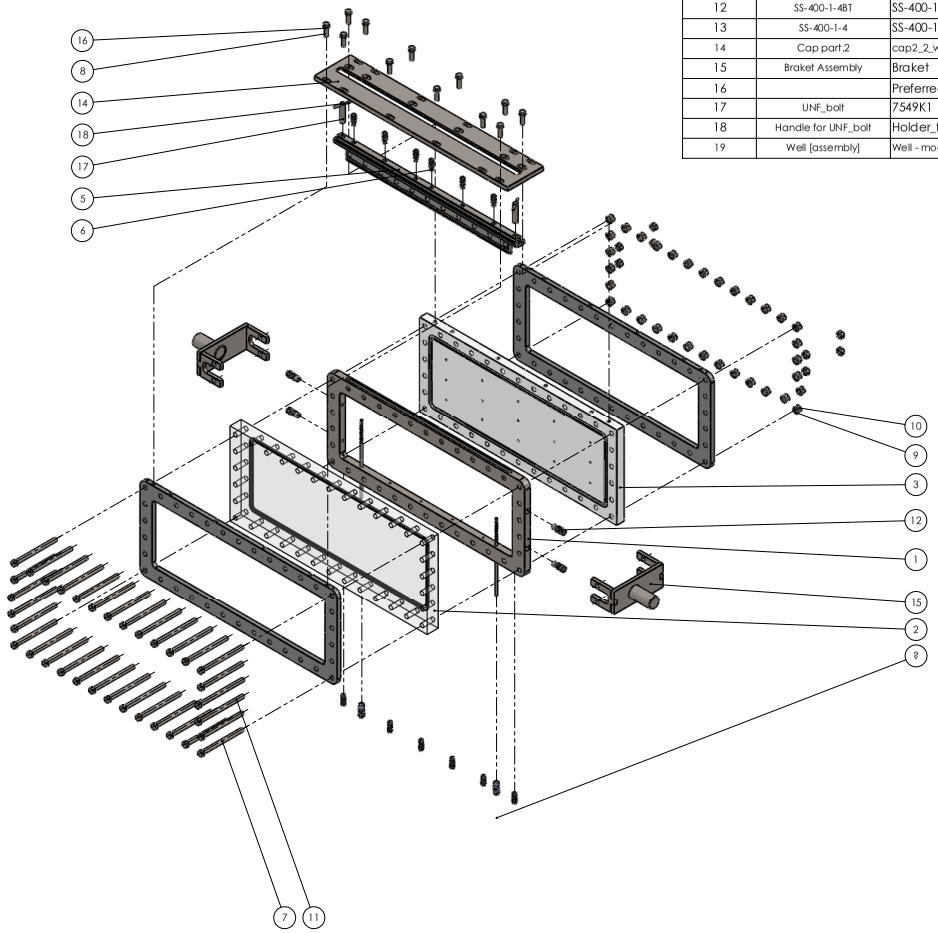


ITEM NO.	SW-Title{Title}	PART NUMBER	Supplier	Material	QTY.
1	MainFrame	MainFrame	MECE Machine shop	AISI 316 Stainless Steel Sheet (SS)	1
2	Transparent side walls	AcrylicSheet_L	MECE Machine shop	Acrylic (Medium-high impact)	1
3	SS-wall with pressure ports	SteelSheet_R_pressureports	MECE Machine shop	AISI 316 Annealed Stainless Steel Bar (SS)	1
4	OutSideFrame	OutSideFrame	MECE Machine shop	Alloy 7075	2
5	Cap part.1	cap1_2	MECE Machine shop	AISI 316 Stainless Steel Sheet (SS)	1
6	SS-200-1-2	SS-200-1-2	Swagelock	AISI 316 Stainless Steel Sheet (SS)	12
7		HBOLT 0.4375-20x5.5x1.125-N			30
8		HBOLT 0.3750-16x1.25x1-N			12
9		Regular LW 0.4375			34
10		HNUT 0.4375-20-D-N			42
11		HBOLT 0.4375-20x6x1.125-N			4
12	SS-400-1-4BT	SS-400-1-4BT	Swagelok Company	AISI 316 Annealed Stainless Steel Bar (SS)	4
13	SS-400-1-4	SS-400-1-4	Swagelok Company	AISI 316 Annealed Stainless Steel Bar (SS)	2
14	Cap part.2	cap2_2_with_slots_modified	MECE Machine shop	Alloy Steel	1
15	Braket Assembly	Braket	MECE Machine shop	Material <not specified>	2
16		Preferred Narrow FW 0.3125			12
17	UNF_bolt	7549K1	MECE Machine shop	Alloy Steel	2
18	Handle for UNF_bolt	Holder_for_7549K1	MECE Machine shop	Alloy Steel	2
19	Well [assembly]	Well - mod_60deg	MECE Machine shop	Material <not specified>	2



Nobes Lab	UNLESS OTHERWISE SPECIFIED:	NAME	The Department of Mechanical Engineering, UNIVERSITY OF ALBERTA
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN: Aleksey Baldygin	TITLE: 2D core holder
email: Baldygin@ualberta.ca	TOLERANCES:	CHECKED: Dr. David Nobes	
Phone: (780)710-5664	ANGULAR: ± 0.5°	ENG APPR.: xxx	
Comments:	.X = ± 0.25	Speed Code: MECE 234	
	.XX = ± 0.05	MecE Job #: xxx	
	.XXX = ± 0.005	File Name: 2DCH_glass_walls	SIZE: C; Project: Quadrise; REV: xxx
	SURFACE FINISH: 0.6 µm	baldygin	
	MATERIAL: XXXXX	12/1/2014 9:47:43 AM	SCALE: 1:5; Mass: xxxx; SHEET 3 OF 11
	Lab Location: NINT 6-93	Monday, December 01, 2014	
	DO NOT SCALE DRAWING		

ITEM NO.	SW-Title(Title)	PART NUMBER	Supplier	Material	QTY.
1	MainFrame	MainFrame	MECE Machine shop	AISI 316 Stainless Steel Sheet (SS)	1
2	Transparent side walls	AcrylicSheet_L	MECE Machine shop	Acrylic (Medium-high impact)	1
3	SS-wall with pressure ports	SteelSheet_R_pressureports	MECE Machine shop	AISI 316 Annealed Stainless Steel Bar (SS)	1
4	OutSideFrame	OutSideFrame	MECE Machine shop	Alloy 7075	2
5	Cap part.1	cap1_2	MECE Machine shop	AISI 316 Stainless Steel Sheet (SS)	1
6	SS-200-1-2	SS-200-1-2	Swagelock	AISI 316 Stainless Steel Sheet (SS)	12
7		HBOLT 0.4375-20x5.5x1.125-N	McMaster-Carr		30
8		HBOLT 0.3750-16x1.25x1-N	McMaster-Carr		12
9		Regular LW 0.4375	McMaster-Carr		34
10		HNUT 0.4375-20-D-N	McMaster-Carr		42
11		HBOLT 0.4375-20x6x1.125-N	McMaster-Carr		4
12	SS-400-1-4BT	SS-400-1-4BT	Swagelok Company	AISI 316 Annealed Stainless Steel Bar (SS)	4
13	SS-400-1-4	SS-400-1-4	Swagelok Company	AISI 316 Annealed Stainless Steel Bar (SS)	2
14	Cap part.2	cap2_2_with_slots_modified	MECE Machine shop	Alloy Steel	1
15	Braket Assembly	Braket	MECE Machine shop	Material <not specified>	2
16		Preferred Narrow FW 0.3125	McMaster-Carr		12
17	UNF_bolt	7549K1	MECE Machine shop	Alloy Steel	2
18	Handle for UNF_bolt	Holder_for_7549K1	MECE Machine shop	Alloy Steel	2
19	Well [assembly]	Well - mod_60deg	MECE Machine shop	Material <not specified>	2



Nobes Lab
Contact: Aleksey Baldygin
Email: Baldygin@ualberta.ca
Phone: (780)710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH: 0.6 µm
MATERIAL: XXXXX
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN	Aleksey Baldygin
	Aleksey Baldygin
CHECKED	Dr. David Nobes
ENG APPR.	xxx
Speed Code	MECE 234
MecE Job #	xxx
File Name:	2DCH_glass_walls

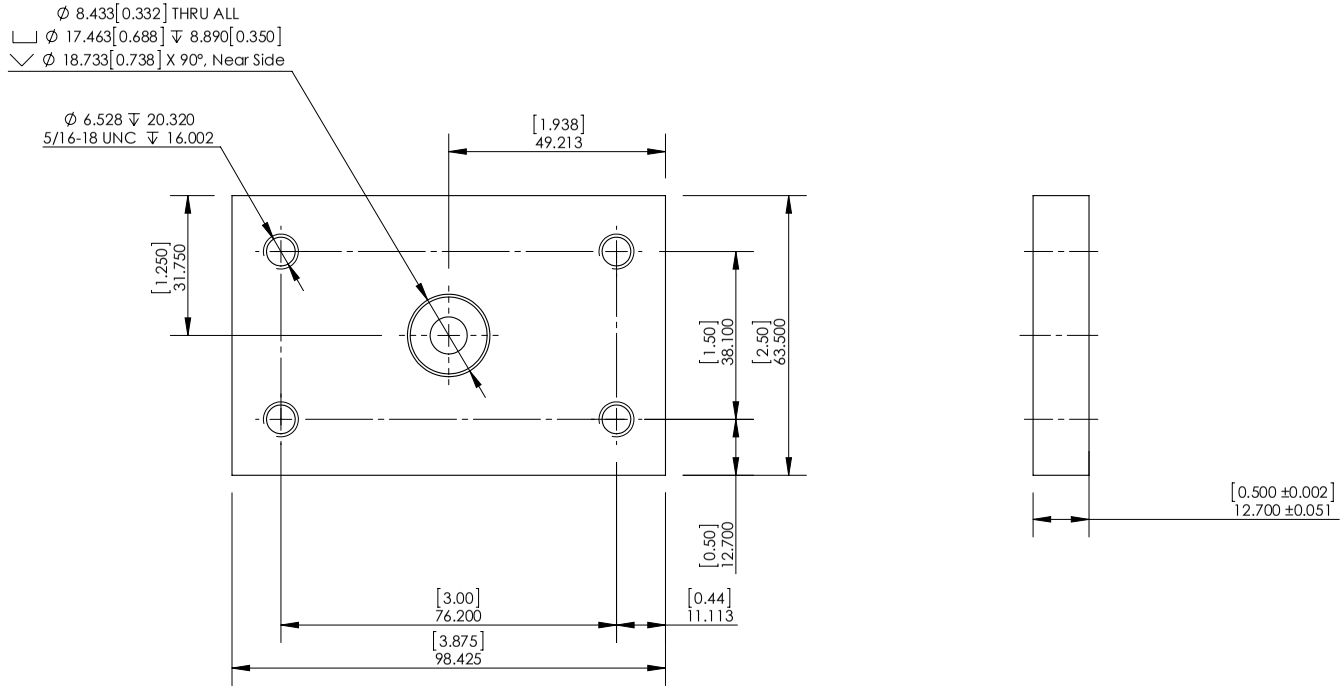
baldygin
12/11/2014 9:47:43 AM
Monday, December 01, 2014

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: 2D core holder

SIZE: C | Project: Quadrise | REV: xxx

SCALE: 1:7 | Mass: xxxxx | SHEET 4 OF 11



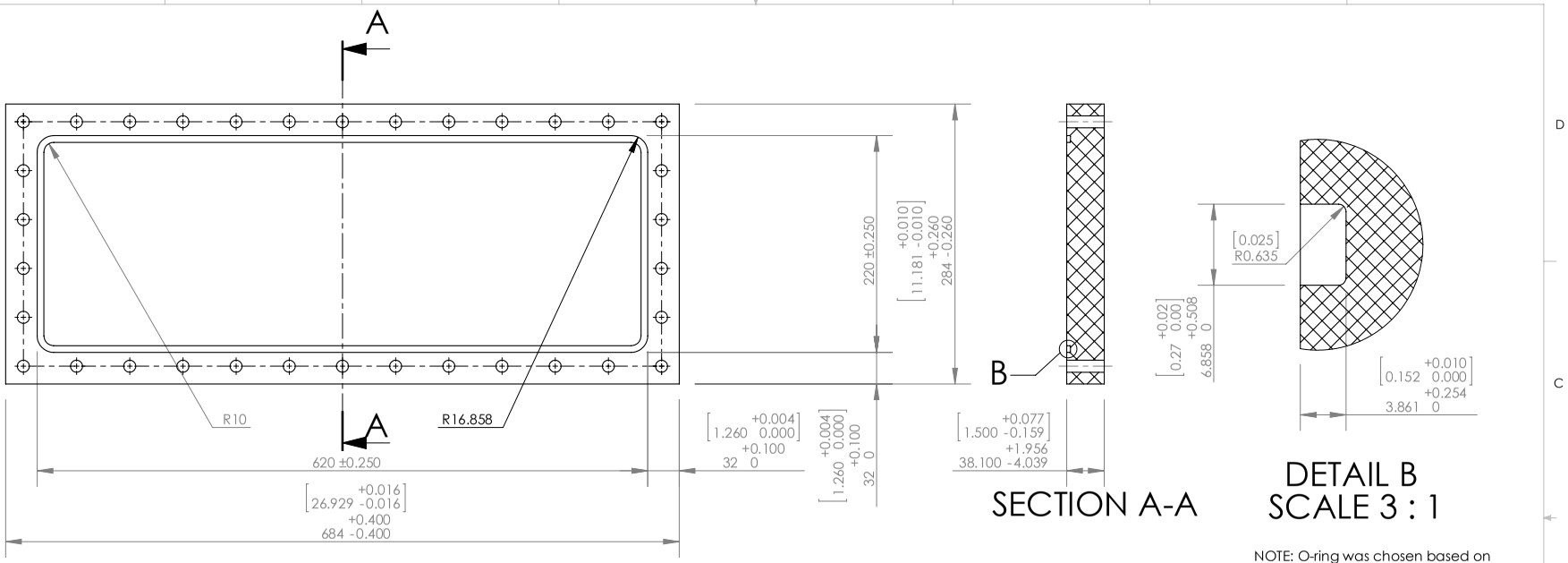
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments:	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: 6061 Alloy	**MecE Job #**	**xxx**
		File Name:	plate for 8020box
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Wednesday, January 08, 2014 12:56:11 PM	
		Monday, November 04, 2013 9:45:45 AM	

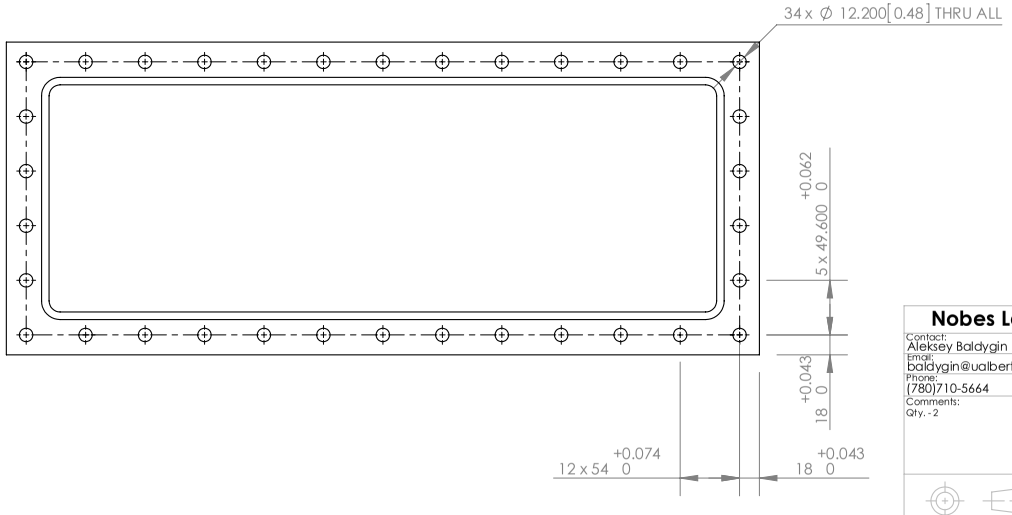
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Plate for 8020 box

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:1	Mass: 0.45	SHEET 5 OF 11





NOTE: O-ring was chosen based on recommendations from Parker Handbook, page 4-18.
Chapter - Face Seal Glands.
Cross section (W) equals to 3/16.

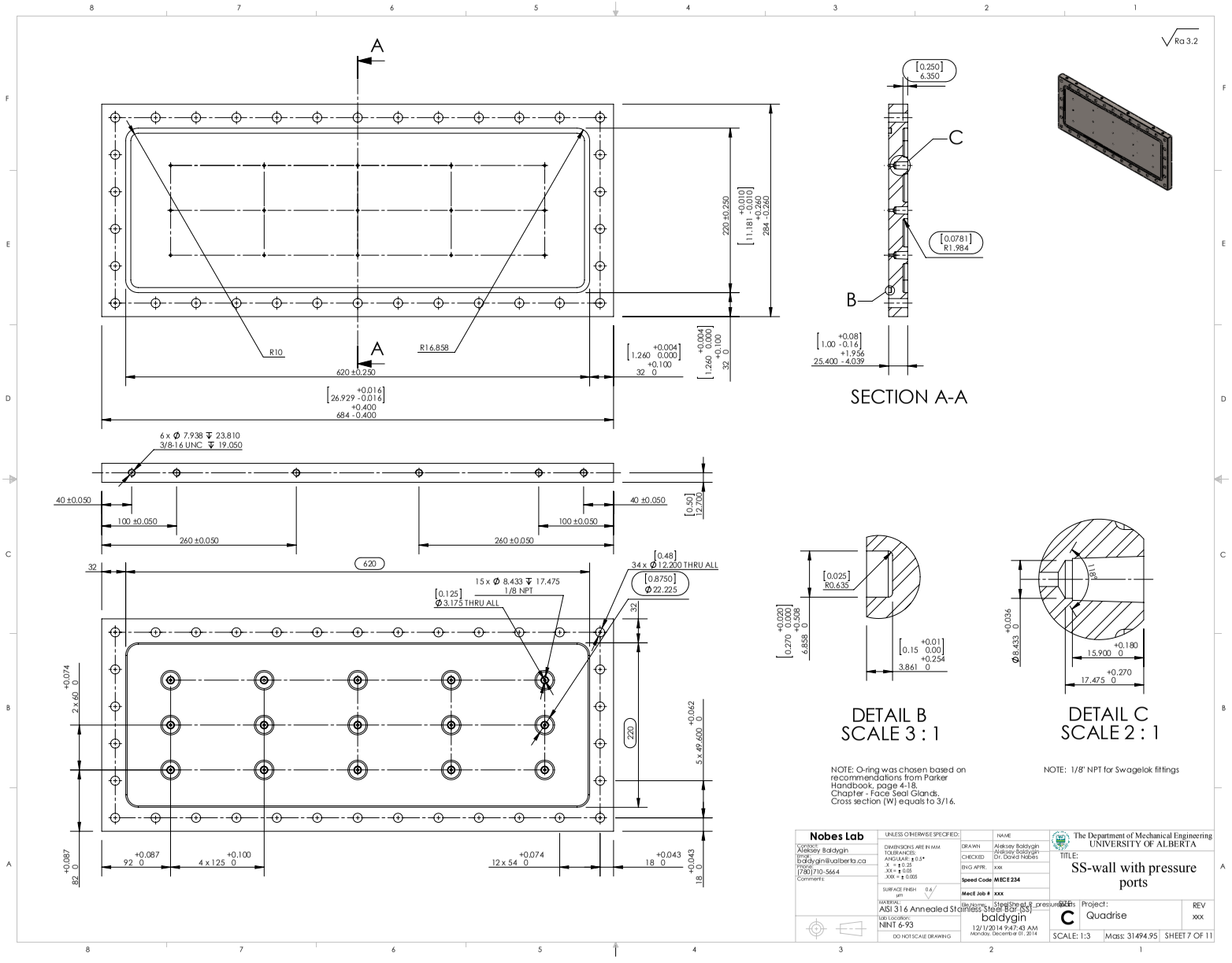
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES:	SMB	Aleksey Baldygin
Phone: (780)710-5664	ANGULAR: ± 0.5°	CHECKED BY	Dr. David Nobes
Comments: Qty. - 2	.X = ± 0.25	**Speed Code**	**MECE 234**
	.XX = ± 0.05	**MecE Job #**	**xxx**
	SURFACE FINISH 0.6 µm	File Name:	AcrylicSheet_L
	MATERIAL: Acrylic (Medium-high impact)	baldygin	
	Lab Location: NINT 6-93	Wednesday, January 08, 2014 2:16:21 PM	
	DO NOT SCALE DRAWING	Friday, May 10, 2013 9:29:30 AM	

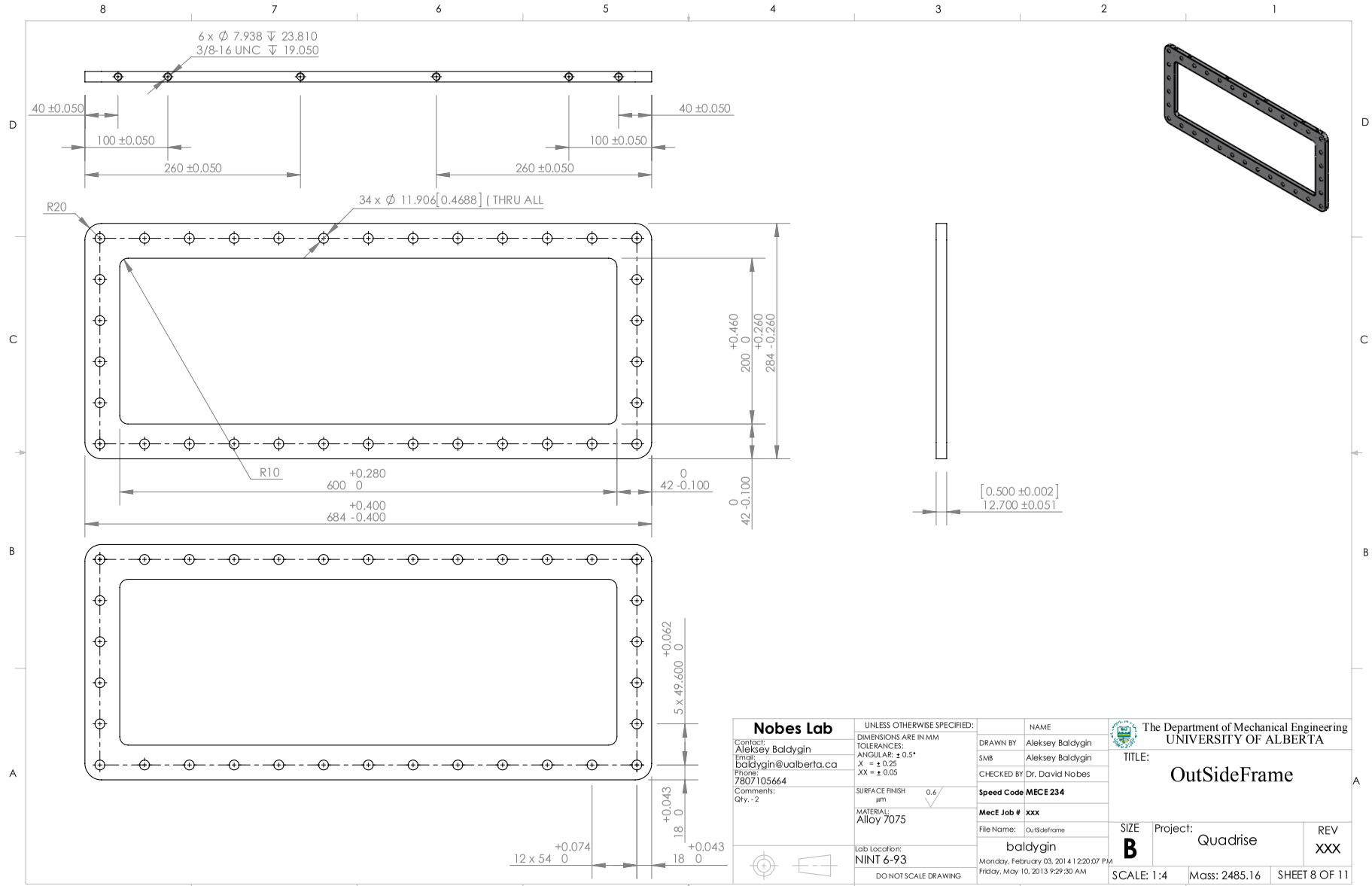
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Transparent side walls

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:4 | Mass: 8648.23 | SHEET 6 OF 11


Ra 3.2
A
A
R10
R16.858
620 ±0.250
[26.929 +0.016 -0.016] 684 +0.400 -0.400
220 ±0.250
[11.181 +0.010 -0.010] 284 +0.260 -0.260
[1.260 +0.004 0.000] 32 +0.100 0
[0.250] 6.350
C
[0.0781] R1.984
B
[1.00 +0.08 -0.16] 25.400 +1.956 -4.039
SECTION A-A
6 x Ø 7.938 ↧ 23.810
3/8-16 UNC ↧ 19.050
40 ±0.050
100 ±0.050
260 ±0.050
[0.50] 12.700
620
32
[0.48] 34 x Ø 12.200 THRU ALL
[0.8750] Ø 22.225
15 x Ø 8.433 ↧ 17.475
1/8 NPT
[0.125] Ø 3.175 THRU ALL
220
2 x 60 +0.074 0
5 x 49.600 +0.062 0
92 +0.087 0
4 x 125 +0.100 0
12 x 54 +0.074 0
18 +0.043 0
[0.025] R0.635
[0.270 +0.020 0.000] 6.858 +0.508 0
[0.15 +0.01 0.00] 3.861 +0.254 0
DETAIL B
SCALE 3 : 1
NOTE: O-ring was chosen based on recommendations from Parker Handbook, page 4-18, Chapter - Face Seal Glands, Cross section (W) equals to 3/16.
3.81
Ø8.433 +0.036 0
15.900 +0.180 0
17.475 +0.270 0
DETAIL C
SCALE 2 : 1
NOTE: 1/8" NPT for Swagelok fittings
Nobes Lab
Contact: Aleksey Baldygin
email: Baldygin@ualberta.ca
Phone: (780)710-5664
Comments:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
.XXX = ± 0.005
SURFACE FINISH µm 0.6
MATERIAL: AISI 316 Annealed Stainless Steel Bar (SS)
DO NOT SCALE DRAWING
NAME
DRAWN Aleksey Baldygin
CHECKED Dr. David Nobes
ENG APPR. xxx
Speed Code MECE 234
MecE Job # xxx
File Name: SteelSheet 8_pressureports
baldygin
12/11/2014 9:47:43 AM
Monday, December 01, 2014
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA
TITLE:
SS-wall with pressure ports
SIZE C
Project: Quadrise
REV xxx
SCALE: 1:3
Mass: 31494.95
SHEET 7 OF 11



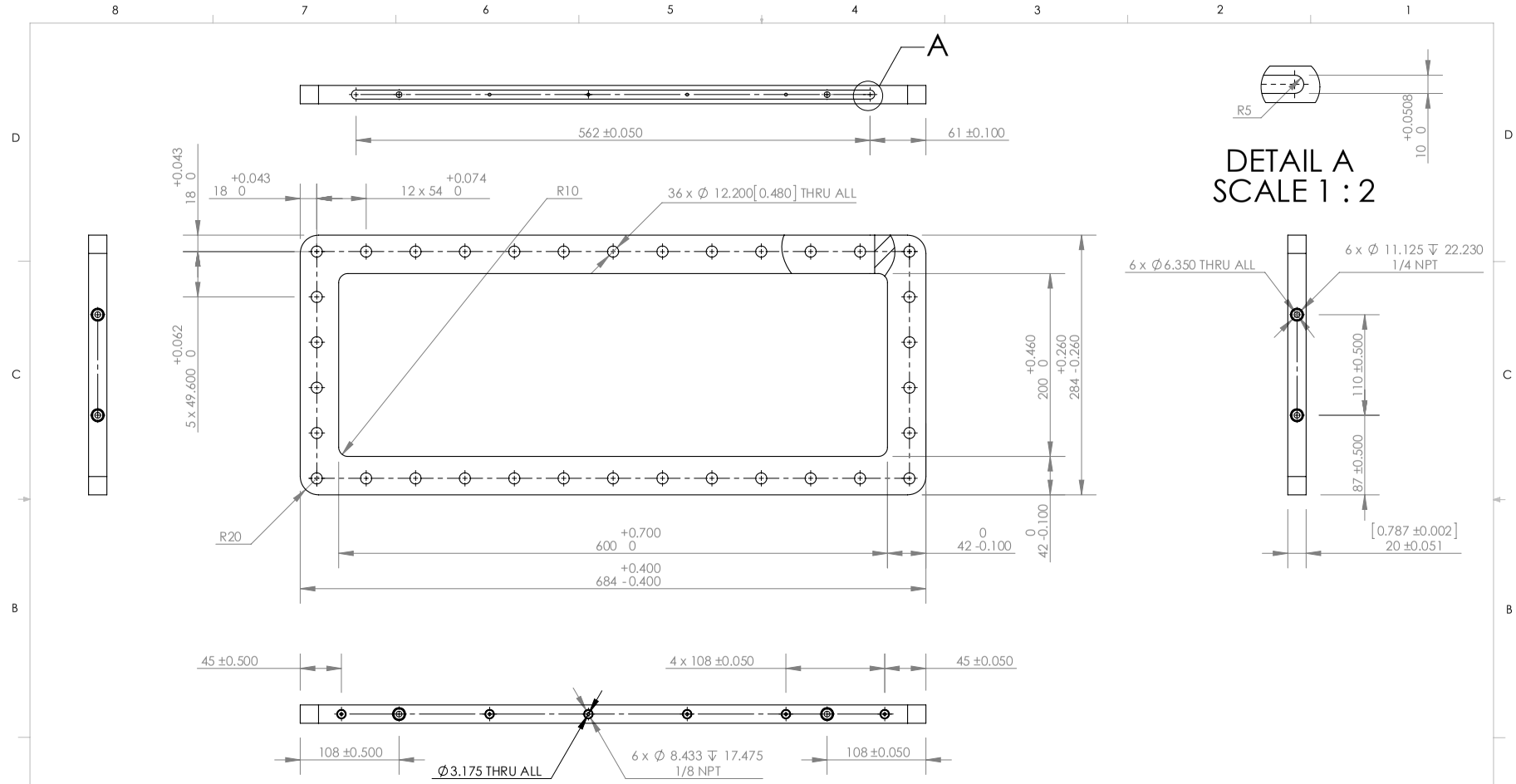
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: 7807105664		CHECKED BY	Dr. David Nobes
Comments: Qty. - 2	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: Alloy 7075	**MecE Job #**	**xxx**
		File Name:	OutSideFrame
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Monday, February 03, 2014 12:20:07 PM Friday, May 10, 2013 9:29:30 AM	

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: OutSideFrame

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:4 | Mass: 2485.16 | SHEET 8 OF 11



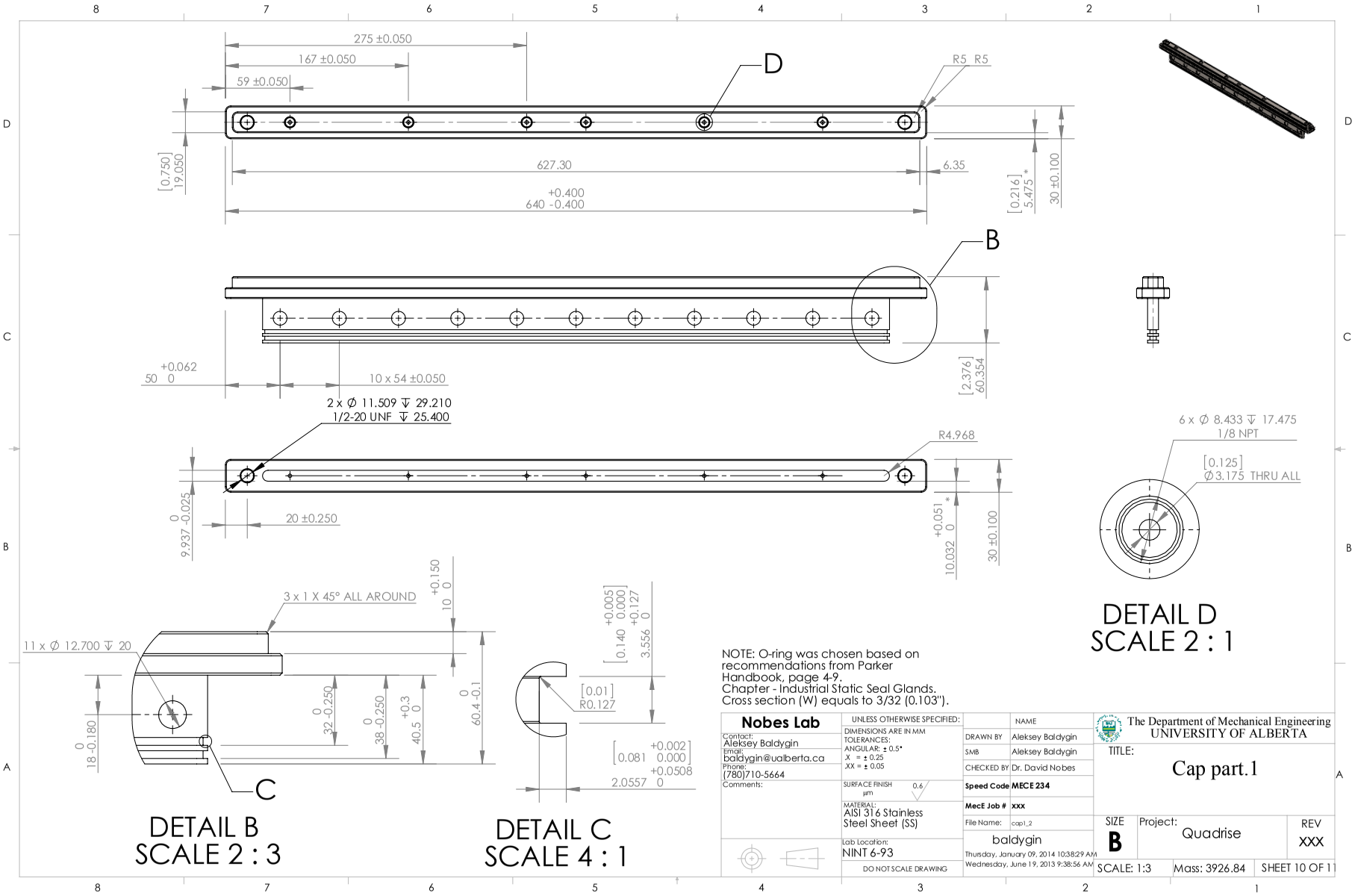
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@uablerta.ca	TOLERANCES:	SMB	Aleksey Baldygin
Phone: (780) 7105664	ANGULAR: ± 0.5°	CHECKED BY	Dr. David Nobes
Comments:	.X = ± 0.25	**Speed Code**	**MECE 234**
	.XX = ± 0.05	**MecE Job #**	**xxx**
	SURFACE FINISH µm 0.6	File Name:	MainFrame
	MATERIAL: AISI 316 Stainless Steel Sheet (SS)	baldygin	
	Lab Location: NINT 6-93	Wednesday, January 08, 2014 3:06:46 PM	
	DO NOT SCALE DRAWING	Friday, May 10, 2013 9:29:30 AM	

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: MainFrame

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:4 | Mass: 9132.49 | SHEET 9 OF 11



DETAIL D
SCALE 2 : 1

DETAIL B
SCALE 2 : 3

DETAIL C
SCALE 4 : 1

NOTE: O-ring was chosen based on recommendations from Parker Handbook, page 4-9.
Chapter - Industrial Static Seal Glands.
Cross section (W) equals to 3/32 (0.103").

Nobes Lab
Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: (780)710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
SURFACE FINISH µm 0.6
MATERIAL: AISI 316 Stainless Steel Sheet (SS)
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN BY	Aleksey Baldygin
SMB	Aleksey Baldygin
CHECKED BY	Dr. David Nobes
Speed Code	MECE 234
MecE Job #	xxx
File Name:	cap1_2

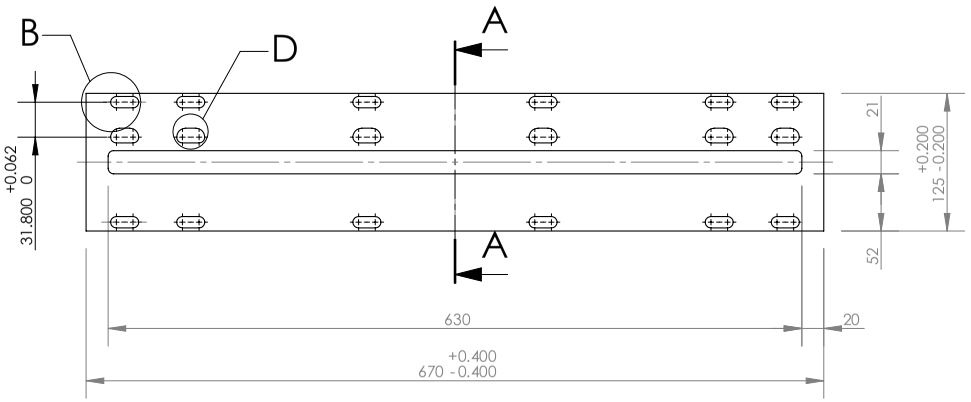
baldygin
Thursday, January 09, 2014 10:38:29 AM
Wednesday, June 19, 2013 9:38:56 AM

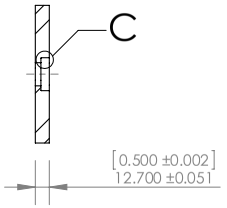
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Cap part.1

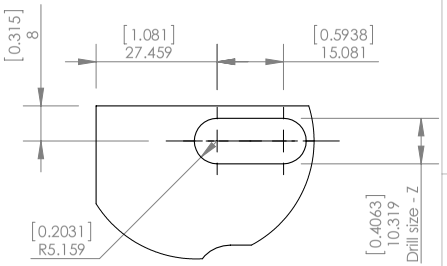
SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:3 | Mass: 3926.84 | SHEET 10 OF 11

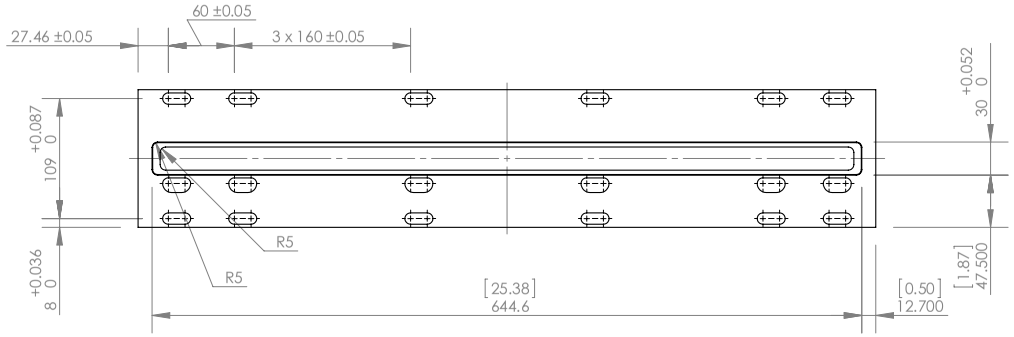


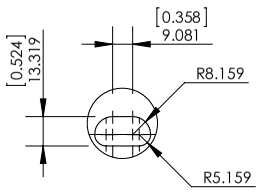


SECTION A-A

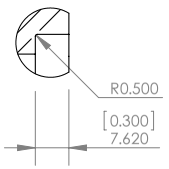


DETAIL B
SCALE 1 : 1





DETAIL D
SCALE 1 : 2



DETAIL C
SCALE 1 : 1

Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments:	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: Alloy Steel	**MecE Job #**	**xxx**
		File Name:	cap2_2_with_slots_modified
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Wednesday, April 30, 2014 11:50:50 AM	
		Wednesday, June 19, 2013 9:38:56 AM	

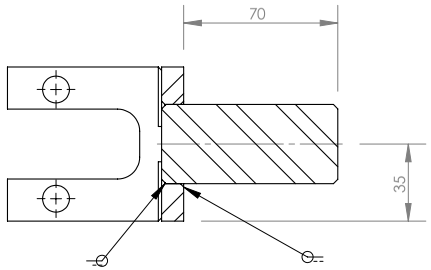
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Cap part.2

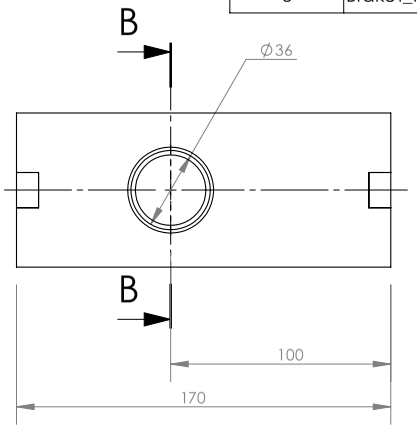
SIZE	Project:	REV
B	Quadrise	XXX

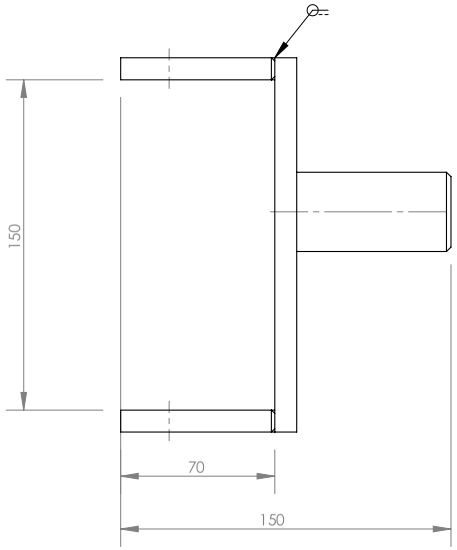
SCALE: 1:4 | Mass: 6084.65 | SHEET 11 OF 11

ITEM NO.	PART NUMBER	Material	QTY.
1	Braket_main	Alloy Steel	1
2	Braket_sidewall	Alloy Steel	2
3	Braket_tube	Alloy Steel	1



SECTION B-B





Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments:	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: Material <not specified>	**MecE Job #**	**xxx**
		File Name:	Braket
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Wednesday, January 08, 2014 4:20:17 PM Monday, October 28, 2013 10:46:15 AM	

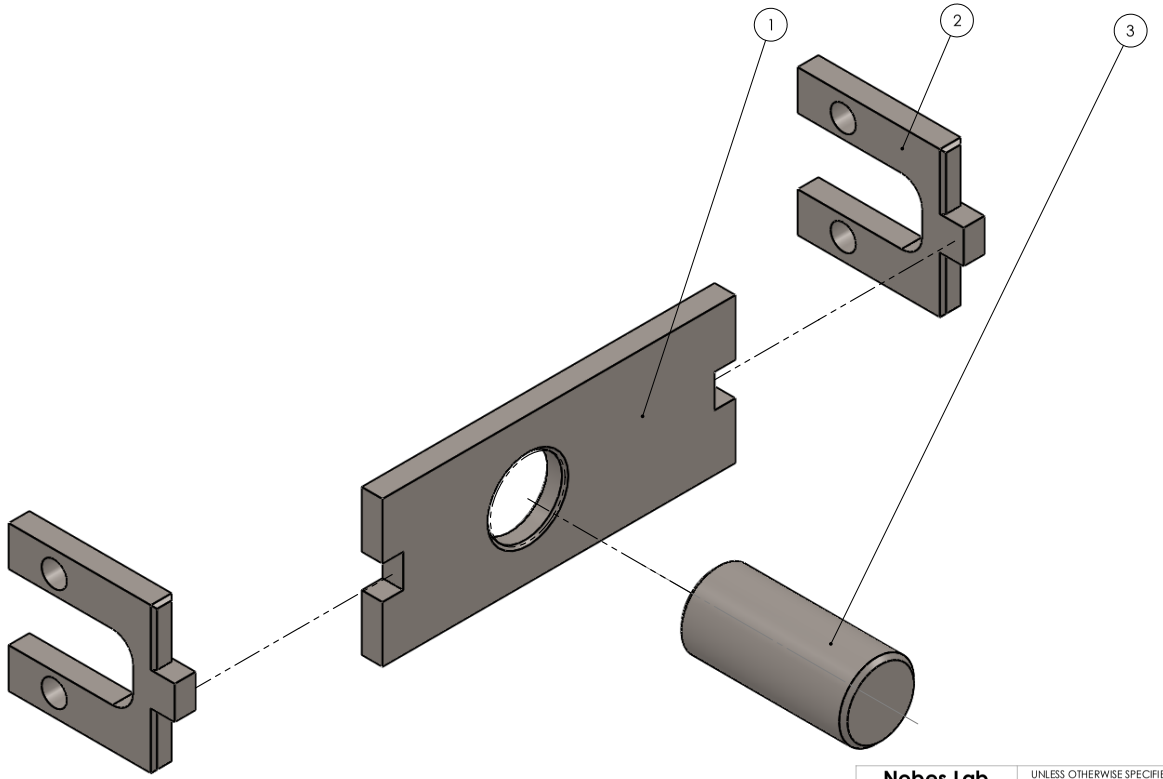
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Braket Assembly

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:2 | Mass: 1889.25 | SHEET 1 OF 5

ITEM NO.	PART NUMBER	Material	QTY.
1	Braket_main	Alloy Steel	1
2	Braket_sidewall	Alloy Steel	2
3	Braket_tube	Alloy Steel	1



Nobes Lab

Contact: Aleksey Baldygin
Email: baldygin@ualberta.ca
Phone: (780)710-5664
Comments:

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
ANGULAR: ± 0.5°
.X = ± 0.25
.XX = ± 0.05
SURFACE FINISH 0.6 µm
MATERIAL: Material <not specified>
Lab Location: NINT 6-93
DO NOT SCALE DRAWING

	NAME
DRAWN BY	
SMB	Aleksey Baldygin
CHECKED BY	Dr. David Nobes
Speed Code	MECE 234
MecE Job #	xxx
File Name:	Braket

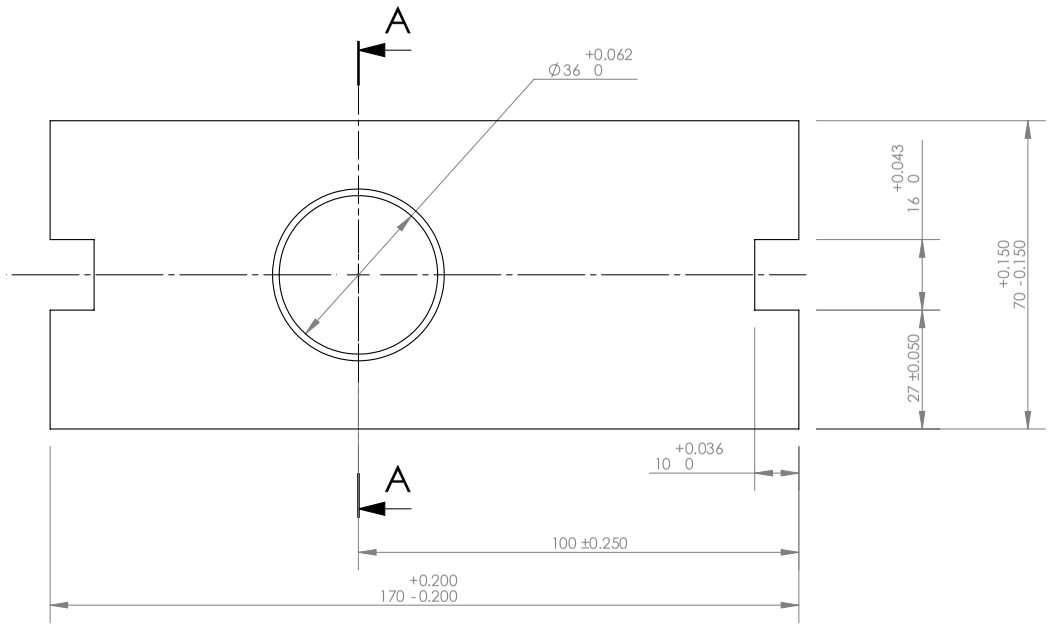
baldygin
Wednesday, January 08, 2014 4:20:17 PM
Monday, October 28, 2013 10:46:15 AM

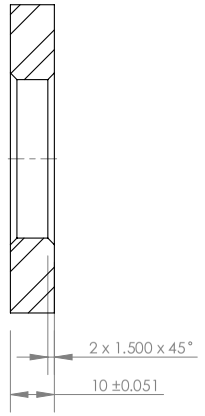
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Braket Assembly

SIZE B | Project: Quadrise | REV xxx

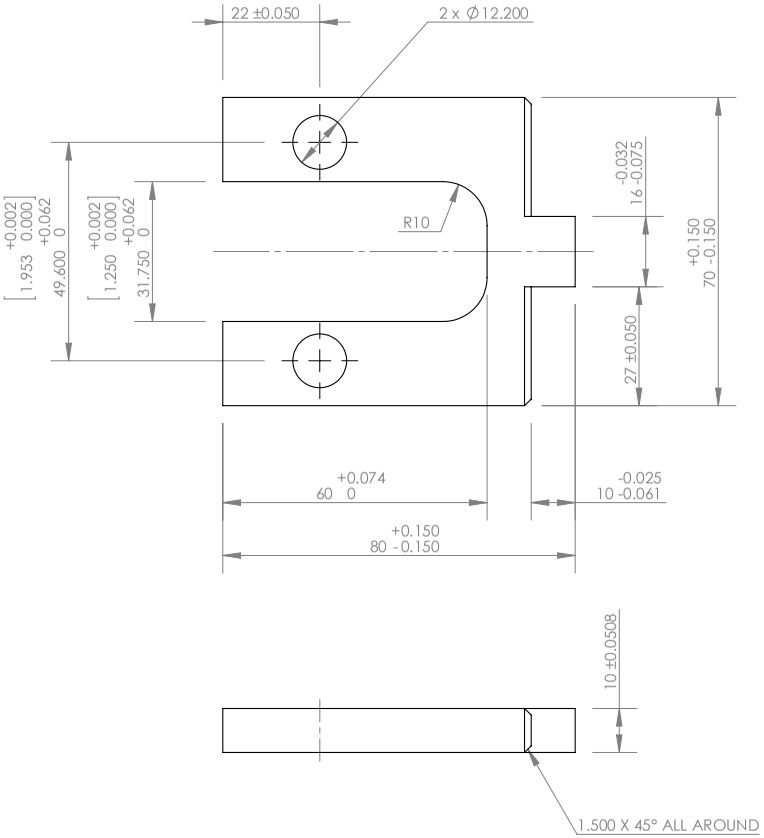
SCALE: 1:1.5 | Mass: 1889.25 | SHEET 2 OF 5

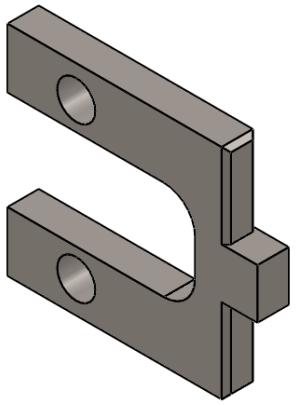




SECTION A-A

Nobes Lab Contact: Aleksey Baldygin Email: baldygin@ualberta.ca Phone: (780)710-5664 Comments: Qty. - 2	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 SURFACE FINISH µm 0.6 MATERIAL: Alloy Steel Lab Location: NINT 6-93 DO NOT SCALE DRAWING	NAME DRAWN BY SMB Aleksey Baldygin CHECKED BY Dr. David Nobes **Speed Code MECE 234** **MecE Job # xxx** File Name: Braket_main baldygin Wednesday, January 08, 2014 12:15:13 PM Wednesday, June 19, 2013 10:50:38 AM	The Department of Mechanical Engineering UNIVERSITY OF ALBERTA TITLE: Braket part.1 SIZE B Project: Quadrise REV xxx SCALE: 1:1 Mass: 811.27 SHEET 3 OF 5





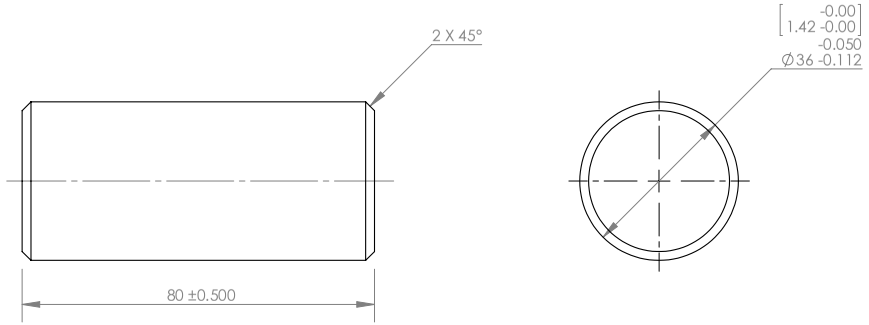
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments: Qty. -2	SURFACE FINISH 0.6 µm	**Speed Code**	**MECE 234**
	MATERIAL: Alloy Steel	**MecE Job #**	**xxx**
		File Name:	Braket_sidewall
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Wednesday, January 08, 2014 12:14:46 PM Wednesday, June 19, 2013 10:50:38 AM	

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Braket part.2

SIZE	Project:	REV
B	Quadrise	XXX

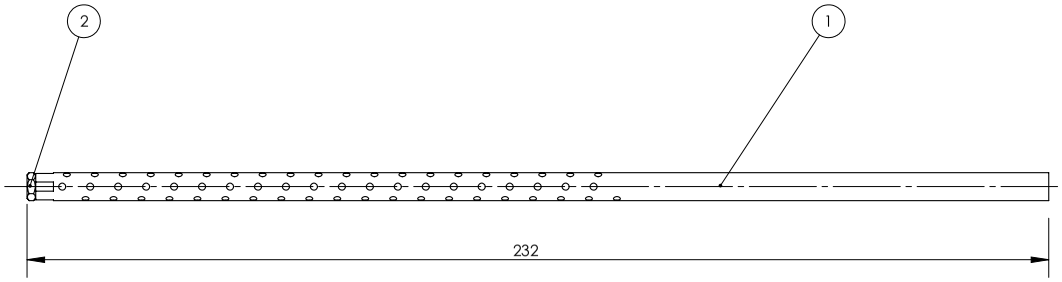
SCALE: 1:1 | Mass: 227.16 | SHEET 4 OF 5

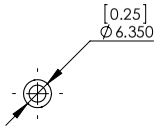


Nobes Lab Contact: Aleksey Baldygin Email: baldygin@ualberta.ca Phone: (780)710-5664 Comments: Qty. - 2	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005 SURFACE FINISH 0.6 µm MATERIAL: Alloy Steel Lab Location: NINT 6-93 DO NOT SCALE DRAWING	NAME DRAWN Aleksey Baldygin CHECKED Dr. David Nobes ENG APPR. xxx Speed Code **MECE 234** MecE Job # **xxx** File Name: Braket_tube baldygin 1/9/2014 11:50:36 AM Friday, September 19, 2014	The Department of Mechanical Engineering UNIVERSITY OF ALBERTA TITLE: Braket part.3 SIZE A | Project: Quadrise | REV xxx SCALE: 1:1 | Mass: 623.66 | SHEET 5 OF 5

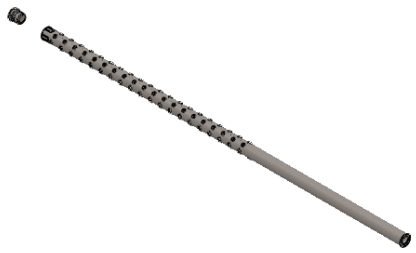
5 4 3 2 1

ITEM NO.	SW-Title(Title)	PART NUMBER	Material	Supplier	QTY.
1	Well [part.1]	maintube_mod_holes	AISI 316 Stainless Steel Sheet (SS)	MECE Machine shop	1
2	Well [part.2]	cap_for_well	AISI 316 Stainless Steel Sheet (SS)	MECE Machine shop	1





Exploded view



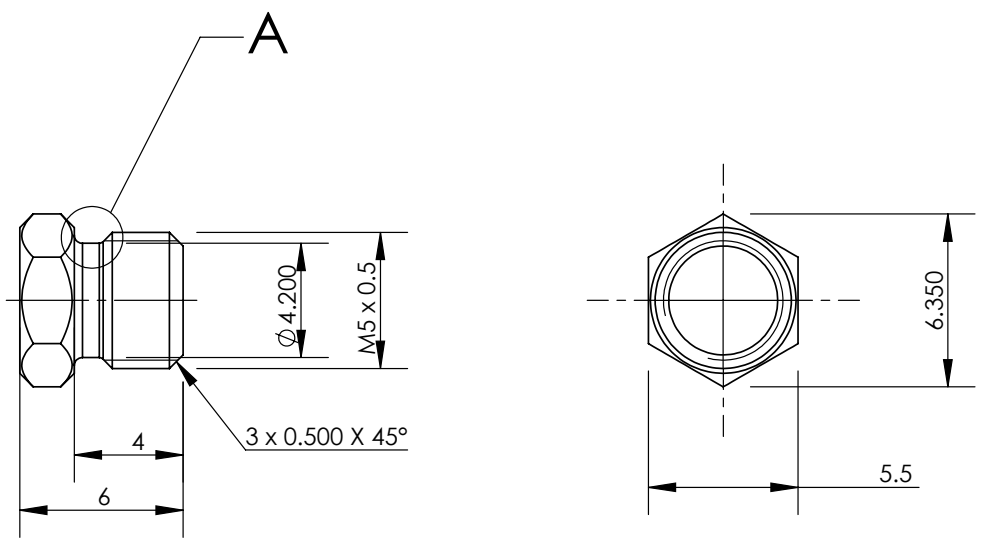
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	Aleksey Baldygin
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments:	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: Material <not specified>	**MecE Job #**	**xxx**
		File Name:	Well
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Friday, September 19, 2014 4:52:13 PM Friday, November 01, 2013 11:02:54 AM	

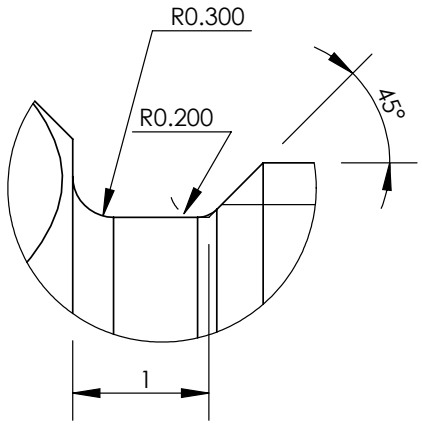
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Well [assembly]

SIZE	Project:	REV
B	Quadrise	XXX

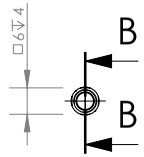
SCALE: 1:1 | Mass: 35.17 | SHEET 1 OF 2

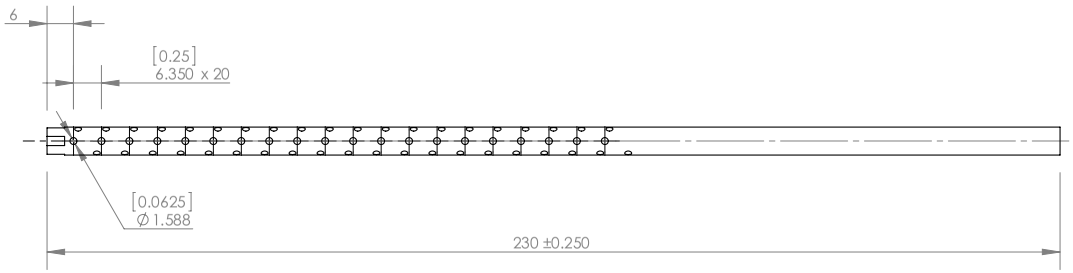


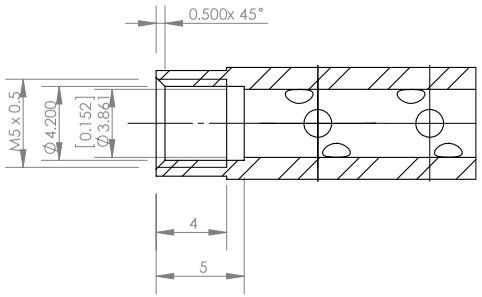


DETAIL A
SCALE 20 : 1

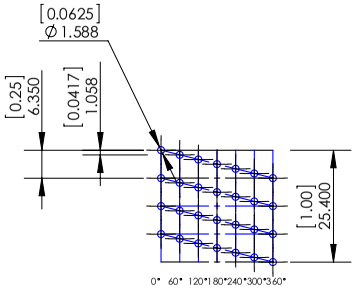
Nobes Lab Contact: Aleksey Baldygin Email: baldygin@ualberta.ca Phone: (780)710-5664 Comments:	UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MM TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05 .XXX = ± 0.005 SURFACE FINISH 0.6 µm MATERIAL: AISI 316 Stainless Steel Sheet (SS) Lab Location: NINT 6-93 DO NOT SCALE DRAWING	NAME DRAWN Aleksey Baldygin CHECKED Aleksey Baldygin Dr. David Nobes ENG APPR. **Speed Code** **MECE 234** **MecE Job #** **xxx** File Name: cap_for_well baldygin 9/19/2014 4:52:13 PM Friday, September 19, 2014	The Department of Mechanical Engineering UNIVERSITY OF ALBERTA TITLE: Well [part.2] SIZE A Project: Quadrise REV xxx SCALE: 4:1 Mass: 0.97 SHEET 2 OF 2







SECTION B-B
SCALE 4 : 1



Hole pattern

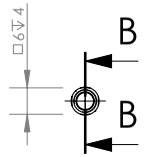
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments: Qty. -2	SURFACE FINISH µm 0.6	**Speed Code**	**MECE 234**
	MATERIAL: AISI 316 Stainless Steel Sheet (SS)	**MecE Job #**	**xxx**
		File Name:	maintube_mod_holes
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Monday, March 24, 2014 10:03:09 PM Friday, November 01, 2013 10:15:04 AM	

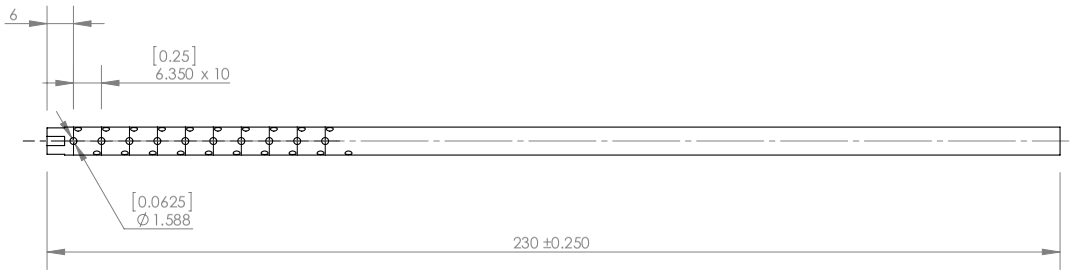
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

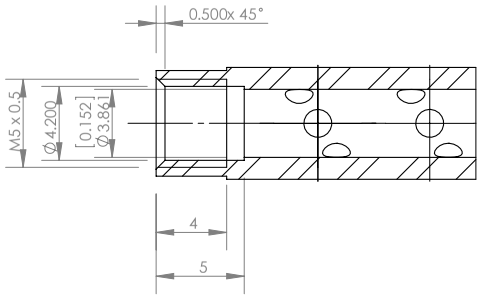
TITLE: Well [part.1]

SIZE	Project:	REV
B	Quadrise	XXX

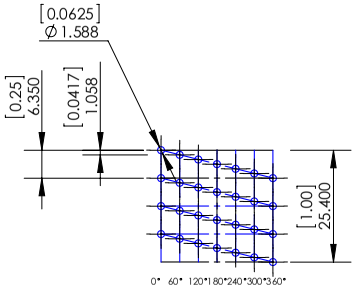
SCALE: 1:1 | Mass: 34.20 | SHEET 1 OF 1







SECTION B-B
SCALE 4 : 1



Hole pattern

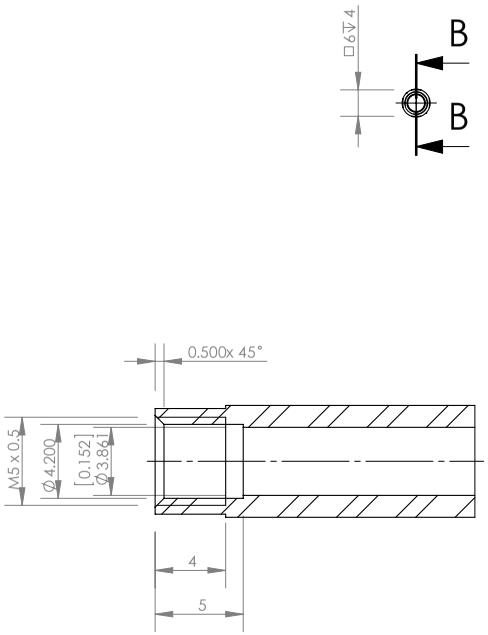
Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments: Qty. -2	SURFACE FINISH 0.6 µm	**Speed Code**	**MECE 234**
	MATERIAL: AISI 316 Stainless Steel Sheet (SS)	**MecE Job #**	**xxx**
		File Name:	maintube_mod_holes_partially
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Tuesday, March 25, 2014 10:23:58 AM Friday, November 01, 2013 10:15:04 AM	

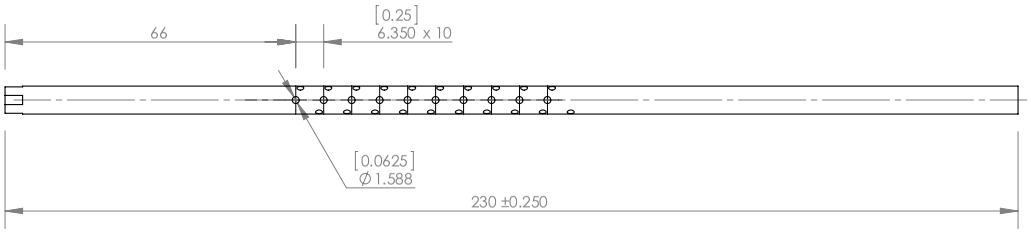
The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Well [part.1]

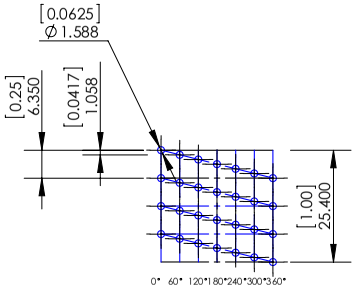
SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:1 | Mass: 35.40 | SHEET 1 OF 1





SECTION B-B
SCALE 4 : 1



Hole pattern

Nobes Lab	UNLESS OTHERWISE SPECIFIED:		NAME
Contact: Aleksey Baldygin	DIMENSIONS ARE IN MM	DRAWN BY	
Email: baldygin@ualberta.ca	TOLERANCES: ANGULAR: ± 0.5° .X = ± 0.25 .XX = ± 0.05	SMB	Aleksey Baldygin
Phone: (780)710-5664		CHECKED BY	Dr. David Nobes
Comments: Qty. -2	SURFACE FINISH 0.6 µm	**Speed Code**	**MECE 234**
	MATERIAL: AISI 316 Stainless Steel Sheet (SS)	**MecE Job #**	**xxx**
		File Name:	maintube_mod_holes_partially
	Lab Location: NINT 6-93	baldygin	
	DO NOT SCALE DRAWING	Tuesday, March 25, 2014 9:32:06 AM	
		Friday, November 01, 2013 9:15:04 AM	

The Department of Mechanical Engineering
UNIVERSITY OF ALBERTA

TITLE: Well [part.1]

SIZE	Project:	REV
B	Quadrise	XXX

SCALE: 1:1	Mass: 35.40	SHEET 1 OF 1

A-7 Software flowchart

The algorithm for the further software was developed and drawn as flowchart using diagramming software(Microsoft Visio Professional, Microsoft, Co.). It was modified and edited along project time, since new features were introduced from experiment to experiment. Final version is shown in Figure A-1.

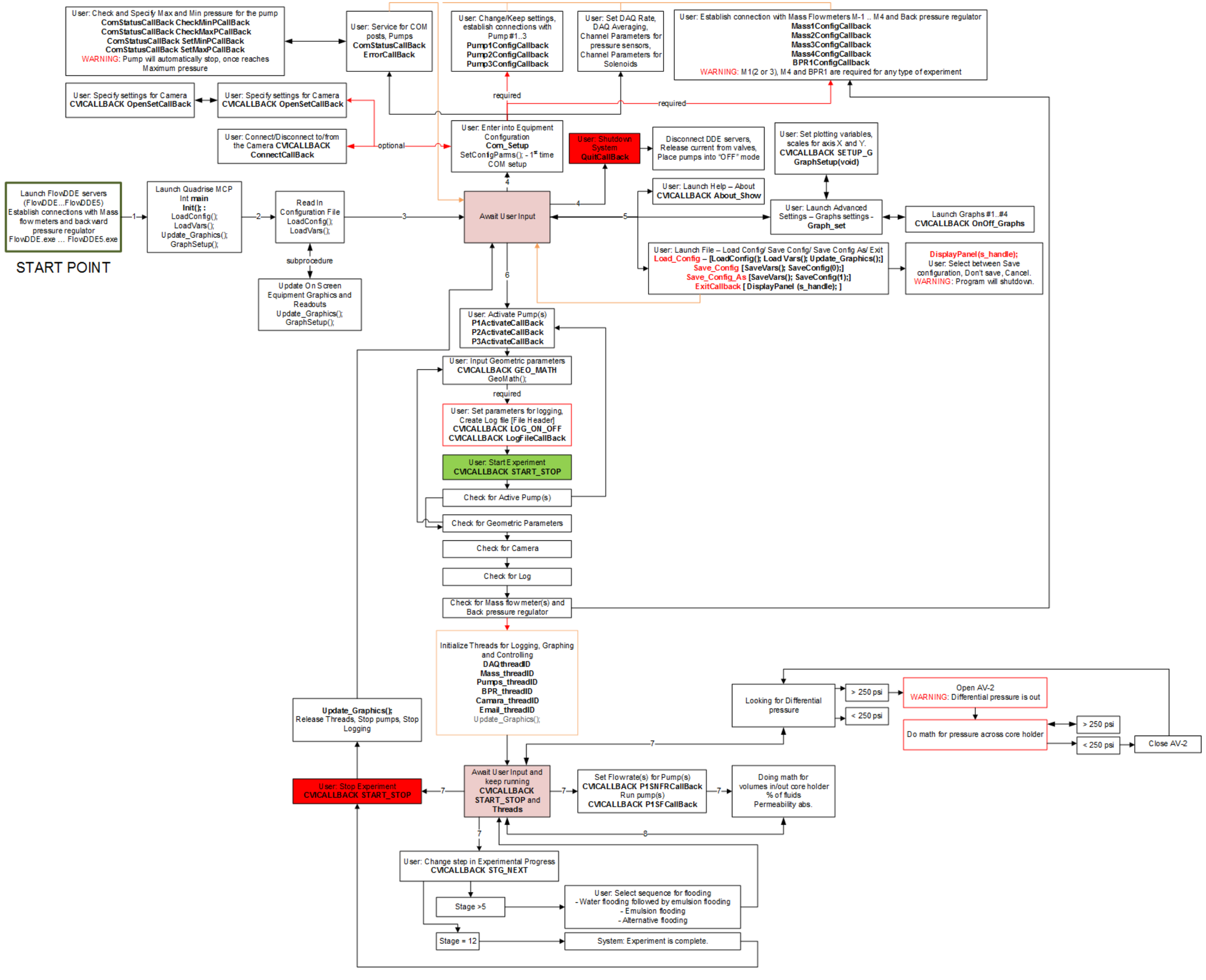


Figure A-1: Software flowchart.

A-8 Software source code

The following code was developed using custom design software (LabWindows CVI 10.0, National Instruments Inc.). The code consists of the five threads which allow to control and collect data from four mass flow meters, three dual piston pumps, 17 inline pressure sensors and one differential pressure sensor, one web-camera and single fraction collector. The code was developed based on source code presented earlier by Marc Evans in his MSc Thesis[3].

[3] Available at Education & Research Archive http://hdl.handle.net/10402/era.28390

```
/*---------------------------------------------------------------------------*/
/* Defines                                                                   */
/*---------------------------------------------------------------------------*/

#define TEXT_LENGTH     2000
#define QuitHelp        1
#define InputqHelp      2
#define NUM 500 //number of samples to hold for plotting

#define subTRUE  1 //used same as dsnTRUE
#define subFALSE 0 //used same as dsnFALSE

#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto Error; else


// IMAQ

#define SOURCE_WINDOW 0
#define DEST_WINDOW 1
#define IM_WIN 0

#define TRUE    1
#define FALSE   0

#define OPEN    1
#define CLOSE   0

#include "nivision.h"                           /* NIVision used to operate different functions of camera */
#include "NIIMAQdx.h"                           /* NIIMAQdx used to operate the camera */

/*---------------------------------------------------------------------------*/
/* Module-globals                                                            */
/*---------------------------------------------------------------------------*/

static int s_handle;      // Handle for Save MCP Config Panel
static int a_handle;      // Handle for About Panel
static int g_handle;      // Handle for Graphs configuration
static int config_handle; // Handle for Configuration system
static int extra_g_handle;// Handle for Extra Graph panel
static int sat_handle;    // Handle for setting volume of water injected
static int panelHandle;   // Handle for camera settings
static int TabPanelHandle; // Handle for tab-settings
static int pressure_handle;
static int pumpservice_handle;

static IMAQdxSession session = 0;
static Image** images = NULL;           // Defining 2-D array variable for video
static Image*  image_g = NULL;          // Defining 1-D array variable for images
static int imageArraySize = 0;
IMAQdxError     error2;

static IMAQdxAttributeInformation* cameraAttributes;
static unsigned int selectedAttrIndex = 0;



int forcestop;

int panel_handle,
    CP,
    LED,
    Just_Opened,
    Just_Grabbed,
    EM_Q, //For emergency stop
    stage,
    pumprun_1,pumprun_2,pumprun_3,

//   Com ports settings
    P1_comport, P2_comport, P3_comport,
    P1_baudrate, P2_baudrate, P3_baudrate,
    P1_portindex, P2_portindex, P3_portindex,
    P1_parity,   P2_parity,   P3_parity,
    P1_databits, P2_databits, P3_databits,
    P1_stopbits, P2_stopbits, P3_stopbits,
    P1_inputq,   P2_inputq,   P3_inputq,         /* Sets input queue length in OpenComConfig */
    P1_outputq,  P2_outputq,  P3_outputq,      /* Sets output queue length in OpenComConfig */
    P1_xmode,    P2_xmode,    P3_xmode,
    P1_ctsmode,  P2_ctsmode,  P3_ctsmode,
    P1_port_open,P2_port_open,P3_port_open,
    com_num,
    p1check, p2check, p3check,

    RS232Error,
    config_flag,
    com_status,
    inqlen,           // Stores result from GetInQLen
    outqlen,          // Stores result from GetOutQLen
    test,             // variable parameter for testing

    M1DDE_connection, M2DDE_connection, M3DDE_connection, M4DDE_connection, BPRDDE_connection,
    m1check, m2check, m3check, m4check, bprcheck,


    stringsize,
    bytes_sent,
    bytes_read,
    read term,
```

```
 98     text_panel_name,  // text inside the panel
 99     PTEXT_panel_name,
100     panel_name;       // panel name
101
102
103   //  breakstatus,
104   //  send_mode,
105   //  send_byte,
106   //  send_term_index,
107   //  read_term_index,
108
109
110 double wb_setpoint,pmax_setpoint,pmin_setpoint;   // set point for flow rate
111
112 int hour_ref, min_ref, sec_ref, hour_cur, min_cur, sec_cur, check_first_time;
113
114
115 //------------------------------------------------------------------------
116 // GRAPH HANDLES
117 int SAMPLES;
118 int OnOff_G1,OnOff_G2,OnOff_G3,OnOff_G4,OnOff_G5,OnOff_G6; //turning graphs On_Off
119
120 // PLOTTING VARIABLES
121 int     plotVar_G_1,plotVar_G_2,plotVar_G_3,plotVar_G_4,plotVar_G_5,plotVar_G_6;
122
123 double  X_Range_G_1,X_Range_G_2,X_Range_G_3,X_Range_G_4;
124
125 int     Y_Mode_G_1,Y_Mode_G_2,Y_Mode_G_3,Y_Mode_G_4,
126         X_Mode_G_5,Y_Mode_G_5;
127 double  Y_Min_G_1,Y_Min_G_2,Y_Min_G_3,Y_Min_G_4,
128         X_Min_G_5,Y_Min_G_5;
129 double  Y_Max_G_1,Y_Max_G_2,Y_Max_G_3,Y_Max_G_4,
130         X_Max_G_5,Y_Max_G_5;
131
132
133 //Core Geometry
134 double L_core, D_core, A_core, V_core, WAT_core, Por_core, Por_core_per, Perm_coef;
135
136
137
138  // LOGGING VARIABLES
139 int Is_P1, Is_P2, Is_P3,
140     Is_M1, Is_M2, Is_M3, Is_M4,
141     Is_BPR,
142     Is_DAQ,
143     Is_Cam,
144     Is_PRES,
145     Is_Log,
146     Is_Graph,
147     Log_File_Selected,
148     Type_exp;
149
150 int globalemail;
151
152 double timeDAQ, timeBPR, timeMFR;
153
154
155 // CONFIG VARIABLES
156 int Exp_Type;
157 // STAGES
158
159 int STG_1, STG_2, STG_3, STG_4, STG_5, STG_6, STG_7, STG_8, STG_9,
160     STG_10, STG_11, STG_12, STG_13, STG_14, STG_15, STG_16;
161
162
163
164 //-------------------------------------------------
165 // Threading COMMUNICATION
166 //-------------------------------------------------
167 // New thread to run the DAQ
168 int CVICALLBACK DAQThreadFunction (void *functionData);
169 static int  DAQthreadID;
170 static int  DAQquitflag = 0;
171 static int  DAQcmtStatus;
172
173 // Thread to get data from Pumps
174 int CVICALLBACK Pumps_ThreadFunction (void *functionData);
175 static int  Pumps_threadID;
176 static int  Pumps_quitflag = 0;
177 static int  Pumps_cmtStatus;
178
179 // Thread to get data from Mass flow meters
180 int CVICALLBACK Mass_ThreadFunction (void *functionData);
181 static int  Mass_threadID;
182 static int  Mass_quitflag = 0;
183 static int  Mass_cmtStatus;
184
185 // Thread to controll and receive data from Backward pressure regulator
186 int CVICALLBACK BPR_ThreadFunction (void *functionData);
187 static int  BPR_threadID;
188 static int  BPR_quitflag = 0;
189 static int  BPR_cmtStatus;
190
191 // Thread to images from camera
192 int CVICALLBACK Camera_ThreadFunction (void *functionData);
193 static int  Camera_threadID;
194 static int  Camera_quitflag = 0;
```

```
195 static int  Camera_cmtStatus;
196
197 // Thread to send email
198 int CVICALLBACK Email_ThreadFunction (void *functionData);
199 static int  Email_threadID;
200 static int  Email_quitflag = 0;
201 static int  Email_cmtStatus;
202
203 CmtThreadLockHandle DAQ_lockHandle, Pumps_lockHandle, Mass_lockHandle, BPR_lockHandle, Camera_lockHandle, Email_lockHandle;
204
205
206 /*---------------------------------------------------------------------------*/
207 //                              DDE client                                   //
208
209 static unsigned int hConv, hConv2, hConv3, hConv4, hConv5;  //DDE Conversation_handle for different servers.
210
211 int DDECallback (unsigned handle, char *topicName,
212                       char *itemName, int xType, int dataFmt,
213                       int dataSize, void *dataPtr,
214                       void *callbackData);
215
216
217 short read_cnt;
218       read_len;
219
220 double P1_timeout, P2_timeout, P3_timeout;
221
222 double viscoef, permcoef;
223
224 char P1_devicename[30],
225      P2_devicename[30],
226      P3_devicename[30],
227      send_data[TEXT_LENGTH],
228      get_send_data[TEXT_LENGTH],
229      read_data[TEXT_LENGTH],
230
231      TP2channel[TEXT_LENGTH],
232
233      DP1channel_A[TEXT_LENGTH],
234      DP1channel_B[TEXT_LENGTH],
235
236      TP1channel[TEXT_LENGTH],
237
238      CP11_Achannel[TEXT_LENGTH],
239      CP11_Bchannel[TEXT_LENGTH],
240      CP12_Achannel[TEXT_LENGTH],
241      CP12_Bchannel[TEXT_LENGTH],
242      CP13_Achannel[TEXT_LENGTH],
243      CP13_Bchannel[TEXT_LENGTH],
244      CP14_Achannel[TEXT_LENGTH],
245      CP14_Bchannel[TEXT_LENGTH],
246      CP15_Achannel[TEXT_LENGTH],
247      CP15_Bchannel[TEXT_LENGTH],
248
249      CP21_Achannel[TEXT_LENGTH],
250      CP21_Bchannel[TEXT_LENGTH],
251      CP22_Achannel[TEXT_LENGTH],
252      CP22_Bchannel[TEXT_LENGTH],
253      CP23_Achannel[TEXT_LENGTH],
254      CP23_Bchannel[TEXT_LENGTH],
255      CP24_Achannel[TEXT_LENGTH],
256      CP24_Bchannel[TEXT_LENGTH],
257      CP25_Achannel[TEXT_LENGTH],
258      CP25_Bchannel[TEXT_LENGTH],
259
260      CP31_Achannel[TEXT_LENGTH],
261      CP31_Bchannel[TEXT_LENGTH],
262      CP32_Achannel[TEXT_LENGTH],
263      CP32_Bchannel[TEXT_LENGTH],
264      CP33_Achannel[TEXT_LENGTH],
265      CP33_Bchannel[TEXT_LENGTH],
266      CP34_Achannel[TEXT_LENGTH],
267      CP34_Bchannel[TEXT_LENGTH],
268      CP35_Achannel[TEXT_LENGTH],
269      CP35_Bchannel[TEXT_LENGTH],
270
271      read_m11data[TEXT_LENGTH],
272      read_m12data[TEXT_LENGTH],
273      read_m13data[TEXT_LENGTH],
274      read_m14data[TEXT_LENGTH],
275
276      read_m21data[TEXT_LENGTH],
277      read_m22data[TEXT_LENGTH],
278      read_m23data[TEXT_LENGTH],
279      read_m24data[TEXT_LENGTH],
280
281      read_m31data[TEXT_LENGTH],
282      read_m32data[TEXT_LENGTH],
283      read_m33data[TEXT_LENGTH],
284      read_m34data[TEXT_LENGTH],
285
286      read_m41data[TEXT_LENGTH],
287      read_m42data[TEXT_LENGTH],
288      read_m43data[TEXT_LENGTH],
289      read_m44data[TEXT_LENGTH],
290
291      read bpr11data[TEXT LENGTH],
```

```
292       read_bpr12data[TEXT_LENGTH],
293       read_bpr13data[TEXT_LENGTH],
294
295       tbox_read_data[TEXT_LENGTH],
296
297       SRP_read_data[TEXT_LENGTH],
298
299       com_msg[500],
300       msg[100];
301
302  char p1frwas[TEXT_LENGTH];
303  char p2frwas[TEXT_LENGTH];
304  char p3frwas[TEXT_LENGTH];
305
306
307  static char cfgfile[300];
308  static char logFile[300];
309  char dirname[MAX_PATHNAME_LEN];
310  char pathname[MAX_PATHNAME_LEN];
311
312  char camName[64];
313
314  //email featur
315  char currentresuts [500];
316
317  CVIAbsoluteTime timestamp;
318  CVIAbsoluteTime timestamp2;
319
320  /*---------------------------------------------------------------------------*/
321  /* DATA LOGGING ARRAYS                                                       */
322  /*---------------------------------------------------------------------------*/
323
324  double step[NUM],
325         bpr_set, bpr_set_was;
326
327  double refwaterinj;
328
329  double //flow rates
330         PFR11[NUM],  //pump #1
331         PFR21[NUM],  //pump #2
332         PFR31[NUM],  //pump #3
333         //pressure
334         P11[NUM],    //pump #1
335         P21[NUM],    //pump #2
336         P31[NUM],    //pump #3
337
338         TP2[NUM],    //inlet
339         DP1[NUM],    //differential
340          DP1_A[NUM],
341          DP1_B[NUM],
342         TP1[NUM],    //outlet
343
344         CP11[NUM],
345          CP11_A[NUM],
346          CP11_B[NUM],
347         CP12[NUM],
348          CP12_A[NUM],
349          CP12_B[NUM],
350         CP13[NUM],
351          CP13_A[NUM],
352          CP13_B[NUM],
353         CP14[NUM],
354          CP14_A[NUM],
355          CP14_B[NUM],
356         CP15[NUM],
357          CP15_A[NUM],
358          CP15_B[NUM],
359
360         CP21[NUM],
361          CP21_A[NUM],
362          CP21_B[NUM],
363         CP22[NUM],
364          CP22_A[NUM],
365          CP22_B[NUM],
366         CP23[NUM],
367          CP23_A[NUM],
368          CP23_B[NUM],
369         CP24[NUM],
370          CP24_A[NUM],
371          CP24_B[NUM],
372         CP25[NUM],
373          CP25_A[NUM],
374          CP25_B[NUM],
375
376         CP31[NUM],
377          CP31_A[NUM],
378          CP31_B[NUM],
379         CP32[NUM],
380          CP32_A[NUM],
381          CP32_B[NUM],
382         CP33[NUM],
383          CP33_A[NUM],
384          CP33_B[NUM],
385         CP34[NUM],
386          CP34_A[NUM],
387          CP34_B[NUM],
388         CP35[NUM],
```

```
         CP35_A[NUM],
         CP35_B[NUM],

//mass #1
        M11[NUM],
        M12[NUM],
        M13[NUM],
        M14[NUM], //extra
//mass #2
        M21[NUM],
        M22[NUM],
        M23[NUM],
        M24[NUM],
//mass #3
        M31[NUM],
        M32[NUM],
        M33[NUM],
        M34[NUM], //extra
//mass #4
        M41[NUM],
        M42[NUM],
        M43[NUM],
        M44[NUM],
// set point out. Mass flow meter
        SETPOINTOUT[NUM],

// OIL AND WATER OUT!
        WATERREC[NUM],
        OILREC[NUM],
        TEMPSTEP[NUM],
        TEMPDELTA[NUM],
        Gtime[NUM],
        Gtime_E[NUM], //current time for emulsion's mass flow meter

        EMULIN[NUM],
        TEMPDELTA_E[NUM],
        TEMPSTEP_E[NUM],

        WATERREC_ST9[NUM],
        OILREC_ST9[NUM],
        MANUAL_TEST_TUBE[NUM],

        WATERFLOODINJ[NUM],

// Emulsion flooding
        RECEMUL[NUM],
        OILREC_ST14[NUM],

// Atrenate emulsion flooding

        RECEMUL_ALT[NUM],
        OILREC_ALT[NUM],

//BPR #1
        BPR11[NUM], //fmeasure
        BPR12[NUM],
        BPR13[NUM],
        BPR14[NUM],
        BPR15[NUM],
        BPR16[NUM],
        BPR17[NUM],
        BPR18[NUM],
        MATRIX[NUM], //extra

        ZEROTP2[NUM],
        ZERODP1[NUM],
        ZEROTP1[NUM],

        ZEROCP11[NUM],
        ZEROCP12[NUM],
        ZEROCP13[NUM],
        ZEROCP14[NUM],
        ZEROCP15[NUM],

        ZEROCP21[NUM],
        ZEROCP22[NUM],
        ZEROCP23[NUM],
        ZEROCP24[NUM],
        ZEROCP25[NUM],

        ZEROCP31[NUM],
        ZEROCP32[NUM],
        ZEROCP33[NUM],
        ZEROCP34[NUM],
        ZEROCP35[NUM],

        G_WATERINJ[NUM],
        G_OILINJ[NUM],
        G_EMULINJ[NUM],

        WATERINJ[NUM],
             WATERINJ2[NUM],
        OILINJ[NUM],
        EMULINJ[NUM],

        WATERINJPV[NUM],
             WATERINJ2PV[NUM],
```

```
        OILINJPV[NUM],
        EMULINJPV[NUM],

        OUTLET[NUM],

        PV[NUM],

        SWI[NUM],

        PERMWATERDP1[NUM],
        PERMOILDP1[NUM],
        PERMEMULDP1[NUM],

        PERMWATERDELTA[NUM],
        PERMOILDELTA[NUM],
        PERMEMULDELTA[NUM],

        BPR1TEMP[NUM],
        Delta[NUM];

double  VIS_WATER, VIS_OIL, VIS_EMUL;

// IMAQmx

int wholeNum, decNum;


/*---------------------------------------------------------------------------*/
/* Internal function prototypes                                              */
/*---------------------------------------------------------------------------*/
void Init(void);            // Init some parameters
void LoadConfig(void);      // Load config variables from a file
void SaveConfig(int type);  // Save config vars to a file
void LoadVars(void);
void SaveVars(void);
void MainPanelQuit(void);   // Main panel Quit
void LogFileHeader(void);
void LogFile(void);
void Update_Graphics(void);
void EMG_STOP(int enable); //Emergency stop mode
void BLINK (void);


void GeoMath(void);

void GraphSetup(void);
void GraphName(int Val, int GRAPH, int L1, int T1, int L2, int T2,
                                   int L3, int T3, int L4, int T4); // Put names on the graphs
void Graph(void);
void Graph_Select(int plotVar, int Panel_Graph);

void DSN_Run_DAQ(void);
void DSN_SHIFT_DAQ(void);
// void DSN_SHIFT_MASS(void);

void DSN_SHIFT_MASS(int instrument);


void DSN_SHIFT_Slow(int instrument);
void SHIFT_M1(void);
void SHIFT_M2(void);
void SHIFT_M3(void);
void SHIFT_M4(void);
void SHIFT_BPR1(void);



void DisplayRS232Error (void);
void DisplayHelp (int);

void EnablePumpPanelControls (int);
void DisplayComStatus (void);

//used already
void SetConfigParms (void);
void GetConfigParms (void);

void COM1_OPEN(void);
void COM2_OPEN(void);
void COM3_OPEN(void);

void ActivatePump1 (void);
void ActivatePump2 (void);
void ActivatePump3 (void);

void SNFRP_1 (void);

void LiquidsInj(void);
void vispermunit(void);
void Perm(void);

void P1_GetConfig (void);
void P2_GetConfig (void);
void P3_GetConfig (void);

void SendReadPump (void);
void SendReadPumpUPGRADE (void); //for pressure recording
```

```
583 void SetPumpFlowrate(void);
584 void CheckPressurePump1 (void);
585 void CheckPressurePump2 (void);
586 void CheckPressurePump3 (void);
587
588 void READ_FLOW_M_1(void);
589 void READ_FLOW_M_2(void);
590 void READ_FLOW_M_3(void);
591 void READ_FLOW_M_4(void);
592 void READ_BPR(void);
593
594 void SWITCH_SOLENOID (void);
595 void SWITCH_FRACTION (int channel1, int channel2, int channel3, int channel4);
596 void ThreadsRun (void);
597 void STOPPUMPS (void);
598 void STOPPUMPSsub (void);
599 void STAGES_1_5 (void);
600 void STAGES_6_10 (void);
601 void STAGES_11_16 (void);
602
603 //Camera
604 void DSN_Init_Image(int panelHandle);              // Initiilize frame for image
605 void DSN_Init_Camera(void);
606 void DeInit_Camera(void);
607 void DisplayError(IMAQdxError error);              // Defining error function DisplayError
608 void UpdateAttributeTab(void);
609 void Capture(void);
610 void SetMaxPressure(void);
611 void SetMinPressure(void);
612
613 int value;
614
615 //static int popHandle;                // Pop-up Panel variable
616
617
618
```

```
/*---------------------------------------------------------------------------*/
/* Include files                                                             */
/*---------------------------------------------------------------------------*/
#include "toolbox.h"
#include <analysis.h>
#include <utility.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "Quadrise MCP.h"
#include "Quadrise MCP_Declare.h"
#include <ddesupp.h>
#include <rs232.h>
#include <formatio.h>
#include <string.h>
#include <NIDAQmx.h>         //cDAQ
#include <DAQmxIOctrl.h>     //cDAQ
#include <nivision.h>                           /* NIVision used to operate different functions of camera */
#include <NIIMAQdx.h>                           /* NIIMAQdx used to operate the camera */

int read_flag=0;
int OVERPRES=1;
// int error, Flag, Manual, Auto;  //used to be before, NIIMAQdx was added
int error, Flag, Manual, Auto;
int pi=3.141593;

// DAQ MX

static Image*  image=NULL;

#define DAQmxErrChk(functionCall) if( DAQmxFailed(error=(functionCall)) ) goto Error; else     //DAQmx

// DAQ Globals
int32       DAQerror=0;
TaskHandle  taskHandle=0;
char        chan[256];
int32       DAQrate;
uInt32      DAQsampsPerChan;
int32       DAQnumRead;
uInt32      DAQnumChannels;
float64     *DAQdata=NULL;
float64     *DAQArray_Out=NULL;
int         DAQlog;
char        DAQerrBuff[2048]={'\0'};
double      DAQoutMean=0;
int         DAQ_Task_Started=0;

/*---------------------------------------------------------------------------*/
/* This is the application's entry-point.                                    */
// main             : MCP main function
/*---------------------------------------------------------------------------*/
int main (int argc, char *argv[])
{
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((panel_handle =      LoadPanel (0, "Quadrise MCP.uir", MAIN)) < 0)
        return -1;

    // Set the panel variables

        config_handle =      LoadPanel (0, "Quadrise MCP.uir", CONFIG);
        s_handle =           LoadPanel (0, "Quadrise MCP.uir", SAVE_con);
        a_handle =           LoadPanel (0, "Quadrise MCP.uir", ABOUT);
        g_handle =           LoadPanel (0, "Quadrise MCP.uir", G_SETUP);
        extra_g_handle =     LoadPanel (0, "Quadrise MCP.uir", EXTRA_G);
        sat_handle     =     LoadPanel (0, "Quadrise MCP.uir", SAT);   //Panel to enter saturation
        panelHandle    =     LoadPanel (0, "Quadrise MCP.uir", PANEL); //Panel for camera settings
        pressure_handle =    LoadPanel (0, "Quadrise MCP.uir", PRESSURE);
        pumpservice_handle = LoadPanel (0, "Quadrise MCP.uir", PUMP_SER);


    //setup DAQmx
        SetCtrlAttribute(config_handle,CONFIG_SOL_DECORATION,ATTR_FRAME_COLOR,VAL_RED);

        NIDAQmx_NewPhysChanDOLineCtrl(config_handle,CONFIG_SOLCHANNEL,1);
        NIDAQmx_NewPhysChanDOLineCtrl(config_handle,CONFIG_SOLCHANNEL_2,1);


    DisplayPanel (panel_handle);
    Init(); //initial initialization for program

    SetSleepPolicy (VAL_SLEEP_MORE); // Use SetSleepPolicy to set the amount your program sleeps when LabWindows/CVI checks for e
vents.
                                     // The default policy is VAL_SLEEP_MORE

    RunUserInterface ();
    DiscardPanel (panel_handle);
    CloseCVIRTE ();
    return 0;
}

//*******************************************************
// Init         : Init some variable and do some setup
// build based on DSN_Init
//*******************************************************
void Init()
{
```

```
// Initialize variables
    Just_Grabbed = 1; //using for first time running
    Just_Opened  = 1; //using for first time running
    Is_Log = 0;
    wholeNum=1;
    decNum=1;


 // Is_Start_Stop = 0;
    bpr_set_was=0;
    SAMPLES = 0;
    OnOff_G1 = 1;
    OnOff_G2 = 1;
    OnOff_G3 = 1;
    OnOff_G4 = 1;
    OnOff_G5 = 1;
    OnOff_G6 = 1;

    SETPOINTOUT[0]=930.0;
    MANUAL_TEST_TUBE[0] = 127;
  // Run some functions
    LoadConfig();      //Load configuration file
    LoadVars();        //Load variables from screen
    Update_Graphics(); //Update graphics for panels, buttons, etc.

    pumprun_1 = 12;
    pumprun_2 = 22;
    pumprun_3 = 32;

   // Do some configuring
    GraphSetup();

return;
}

void LoadConfig(void)  //build based on DSN_Load_Config(void)
{
char keyword[50];
char value [50];
FILE *fp;

if(Just_Opened == subTRUE)
{

        cfgfile[0] = '\0';
        strcat(cfgfile, "Quadrise MCP.cfg");

// Set the path to the application directory
GetProjectDir (dirname);
MakePathname (dirname, cfgfile, pathname);
strcpy (cfgfile, pathname);
        Just_Opened = 0; //second recall of Load Config will know that it's second attempt
}
else
{
//selecting the file to save to
FileSelectPopup ("", "*.cfg", "",
                 "Enter the name of the file to LOAD",
                 VAL_LOAD_BUTTON, 0, 0, 1, 0, cfgfile);
}

//opening the file
fp = fopen( cfgfile, "r" );
   if ( fp == NULL )
   {
      MessagePopup ("Config ERROR", "Could not find the config file");
      return;
   }

while( !feof( fp ) )
    {

    fscanf( fp, "%s\t%s\n", keyword, value );

    //Camera
    if ( !strcmp( keyword, "Camera_Name" ) )
        strcpy(camName,value);

//COM PANEL
    //COM 1 [Pump#1]
    else if ( !strcmp( keyword, "P1_comport" ) )
        P1_comport = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_baudrate" ) )
        P1_baudrate = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_parity" ) )
        P1_parity = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_databits" ) )
        P1_databits = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_stopbits" ) )
        P1_stopbits = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_inputq" ) )
        P1_inputq = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_outputq" ) )
        P1_outputq = (int)atoi( value );
    else if ( !strcmp( keyword, "P1_ctsmode" ) )
        P1_ctsmode = (int)atoi( value );
    else if ( !strcmp( keyword, "P1 xmode" ) )
```

```
194         P1_xmode = (int)atoi( value );
195     else if ( !strcmp( keyword, "P1_timeout" ) )
196         P1_timeout = (double)atoi( value );
197     else if ( !strcmp( keyword, "P1_portindex" ) )
198         P1_portindex = (int)atoi( value );
199
200     //COM 2 [Pump#2]
201     else if ( !strcmp( keyword, "P2_comport" ) )
202         P2_comport = (int)atoi( value );
203     else if ( !strcmp( keyword, "P2_baudrate" ) )
204         P2_baudrate = (int)atoi( value );
205     else if ( !strcmp( keyword, "P2_parity" ) )
206         P2_parity = (int)atoi( value );
207     else if ( !strcmp( keyword, "P2_databits" ) )
208         P2_databits = (int)atoi( value );
209     else if ( !strcmp( keyword, "P2_stopbits" ) )
210         P2_stopbits = (int)atoi( value );
211     else if ( !strcmp( keyword, "P2_inputq" ) )
212         P2_inputq = (int)atoi( value );
213     else if ( !strcmp( keyword, "P2_outputq" ) )
214         P2_outputq = (int)atoi( value );
215     else if ( !strcmp( keyword, "P2_ctsmode" ) )
216         P2_ctsmode = (int)atoi( value );
217     else if ( !strcmp( keyword, "P2_xmode" ) )
218         P2_xmode = (int)atoi( value );
219     else if ( !strcmp( keyword, "P2_timeout" ) )
220         P2_timeout = (double)atoi( value );
221     else if ( !strcmp( keyword, "P2_portindex" ) )
222         P2_portindex = (int)atoi( value );
223
224     //COM 3 [Pump#3]
225     else if ( !strcmp( keyword, "P3_comport" ) )
226         P3_comport = (int)atoi( value );
227     else if ( !strcmp( keyword, "P3_baudrate" ) )
228         P3_baudrate = (int)atoi( value );
229     else if ( !strcmp( keyword, "P3_parity" ) )
230         P3_parity = (int)atoi( value );
231     else if ( !strcmp( keyword, "P3_databits" ) )
232         P3_databits = (int)atoi( value );
233     else if ( !strcmp( keyword, "P3_stopbits" ) )
234         P3_stopbits = (int)atoi( value );
235     else if ( !strcmp( keyword, "P3_inputq" ) )
236         P3_inputq = (int)atoi( value );
237     else if ( !strcmp( keyword, "P3_outputq" ) )
238         P3_outputq = (int)atoi( value );
239     else if ( !strcmp( keyword, "P3_ctsmode" ) )
240         P3_ctsmode = (int)atoi( value );
241     else if ( !strcmp( keyword, "P3_xmode" ) )
242         P3_xmode = (int)atoi( value );
243     else if ( !strcmp( keyword, "P3_timeout" ) )
244         P3_timeout = (double)atoi( value );
245     else if ( !strcmp( keyword, "P3_portindex" ) )
246         P3_portindex = (int)atoi( value );
247
248 // GRAPH PANEL
249     // Graph #1
250     else if ( !strcmp( keyword, "plotVar_G_1" ) )
251         plotVar_G_1 =   (int)atoi( value );
252     else if ( !strcmp( keyword, "X_Range_G_1" ) )
253         X_Range_G_1 = (float)atof( value );
254     else if ( !strcmp( keyword, "Y_Mode_G_1" ) )
255         Y_Mode_G_1 =   (int)atoi( value );
256     else if ( !strcmp( keyword, "Y_Min_G_1" ) )
257         Y_Min_G_1 = (float)atof( value );
258     else if ( !strcmp( keyword, "Y_Max_G_1" ) )
259         Y_Max_G_1 = (float)atof( value );
260     // Graph #2
261     else if ( !strcmp( keyword, "plotVar_G_2" ) )
262         plotVar_G_2 =   (int)atoi( value );
263     else if ( !strcmp( keyword, "X_Range_G_2" ) )
264         X_Range_G_2 = (float)atof( value );
265     else if ( !strcmp( keyword, "Y_Mode_G_2" ) )
266         Y_Mode_G_2 =   (int)atoi( value );
267     else if ( !strcmp( keyword, "Y_Min_G_2" ) )
268         Y_Min_G_2 = (float)atof( value );
269     else if ( !strcmp( keyword, "Y_Max_G_2" ) )
270         Y_Max_G_2 = (float)atof( value );
271     // Graph #3
272     else if ( !strcmp( keyword, "plotVar_G_3" ) )
273         plotVar_G_3 =   (int)atoi( value );
274     else if ( !strcmp( keyword, "X_Range_G_3" ) )
275         X_Range_G_3 = (float)atof( value );
276     else if ( !strcmp( keyword, "Y_Mode_G_3" ) )
277         Y_Mode_G_3 =   (int)atoi( value );
278     else if ( !strcmp( keyword, "Y_Min_G_3" ) )
279         Y_Min_G_3 = (float)atof( value );
280     else if ( !strcmp( keyword, "Y_Max_G_3" ) )
281         Y_Max_G_3 = (float)atof( value );
282     // Graph #4
283     else if ( !strcmp( keyword, "plotVar_G_4" ) )
284         plotVar_G_4 =   (int)atoi( value );
285     else if ( !strcmp( keyword, "X_Range_G_4" ) )
286         X_Range_G_4 = (float)atof( value );
287     else if ( !strcmp( keyword, "Y_Mode_G_4" ) )
288         Y_Mode_G_4 =   (int)atoi( value );
289     else if ( !strcmp( keyword, "Y_Min_G_4" ) )
290         Y_Min_G_4 = (float)atof( value );
```

```
291     else if ( !strcmp( keyword, "Y_Max_G_4" ) )
292         Y_Max_G_4 = (float)atof( value );
293
294     // Graph #5
295     else if ( !strcmp( keyword, "plotVar_G_5" ) )
296         plotVar_G_5 =   (int)atoi( value );
297     else if ( !strcmp( keyword, "X_Mode_G_5" ) )
298         X_Mode_G_5 =   (int)atoi( value );
299     else if ( !strcmp( keyword, "X_Min_G_5" ) )
300         X_Min_G_5 = (float)atof( value );
301     else if ( !strcmp( keyword, "X_Max_G_5" ) )
302         X_Max_G_5 = (float)atof( value );
303     else if ( !strcmp( keyword, "Y_Mode_G_5" ) )
304         Y_Mode_G_5 =   (int)atoi( value );
305     else if ( !strcmp( keyword, "Y_Min_G_5" ) )
306         Y_Min_G_5 = (float)atof( value );
307     else if ( !strcmp( keyword, "Y_Max_G_5" ) )
308         Y_Max_G_5 = (float)atof( value );
309
310
311     // Equipment Setup
312     else if ( !strcmp( keyword, "Exp_Type" ) )
313         Exp_Type = (int)atoi( value );
314
315     //Logging Variables
316     else if ( !strcmp( keyword, "Is_P1" ) )
317         Is_P1 = (int)atoi( value );
318     else if ( !strcmp( keyword, "Is_P2" ) )
319         Is_P2 = (int)atoi( value );
320     else if ( !strcmp( keyword, "Is_P3" ) )
321         Is_P3 = (int)atoi( value );
322     else if ( !strcmp( keyword, "Is_M1" ) )
323         Is_M1 = (int)atoi( value );
324     else if ( !strcmp( keyword, "Is_M2" ) )
325         Is_M2 = (int)atoi( value );
326     else if ( !strcmp( keyword, "Is_M3" ) )
327         Is_M3 = (int)atoi( value );
328     else if ( !strcmp( keyword, "Is_M4" ) )
329         Is_M4 = (int)atoi( value );
330     else if ( !strcmp( keyword, "Is_PRES" ) )
331         Is_PRES = (int)atoi( value );
332     else if ( !strcmp( keyword, "Is_BPR" ) )
333         Is_BPR = (int)atoi( value );
334     else if ( !strcmp( keyword, "Is_Graph" ) )
335         Is_Graph = (int)atoi( value );
336     else if ( !strcmp( keyword, "Is_DAQ" ) )
337         Is_DAQ = (int)atoi( value );
338     else if ( !strcmp( keyword, "Is_Log" ) )
339         Is_Log = (int)atoi( value );
340
341     //pressure reading error
342     else if ( !strcmp(keyword, "ZEROTP2" ) )
343         ZEROTP2[0] = (double)atof (value);
344     else if ( !strcmp(keyword, "ZERODP1" ) )
345         ZERODP1[0] = (double)atof (value);
346     else if ( !strcmp(keyword, "ZEROTP1" ) )
347         ZEROTP1[0] = (double)atof (value);
348
349     else if ( !strcmp(keyword, "ZEROCP11" ) )
350         ZEROCP11[0] = (double)atof (value);
351     else if ( !strcmp(keyword, "ZEROCP12" ) )
352         ZEROCP12[0] = (double)atof (value);
353     else if ( !strcmp(keyword, "ZEROCP13" ) )
354         ZEROCP13[0] = (double)atof (value);
355     else if ( !strcmp(keyword, "ZEROCP14" ) )
356         ZEROCP14[0] = (double)atof (value);
357     else if ( !strcmp(keyword, "ZEROCP15" ) )
358         ZEROCP15[0] = (double)atof (value);
359
360     else if ( !strcmp(keyword, "ZEROCP21" ) )
361         ZEROCP21[0] = (double)atof (value);
362     else if ( !strcmp(keyword, "ZEROCP22" ) )
363         ZEROCP22[0] = (double)atof (value);
364     else if ( !strcmp(keyword, "ZEROCP23" ) )
365         ZEROCP23[0] = (double)atof (value);
366     else if ( !strcmp(keyword, "ZEROCP24" ) )
367         ZEROCP24[0] = (double)atof (value);
368     else if ( !strcmp(keyword, "ZEROCP25" ) )
369         ZEROCP25[0] = (double)atof (value);
370
371     else if ( !strcmp(keyword, "ZEROCP31" ) )
372         ZEROCP31[0] = (double)atof (value);
373     else if ( !strcmp(keyword, "ZEROCP32" ) )
374         ZEROCP32[0] = (double)atof (value);
375     else if ( !strcmp(keyword, "ZEROCP33" ) )
376         ZEROCP33[0] = (double)atof (value);
377     else if ( !strcmp(keyword, "ZEROCP34" ) )
378         ZEROCP34[0] = (double)atof (value);
379     else if ( !strcmp(keyword, "ZEROCP35" ) )
380         ZEROCP35[0] = (double)atof (value);
381
382
383
384     //restore last injection
385     else if ( !strcmp(keyword, "G_WATERINJ" ) )
386         G_WATERINJ[0] = (double)atof (value);
387     else if ( !strcmp(keyword, "G OILINJ" ) )
```

```
388         G_OILINJ[0] =  (double)atof (value);
389     else if ( !strcmp(keyword, "G_EMULINJ" ) )
390         G_EMULINJ[0] = (double)atof (value);
391
392     else if ( !strcmp(keyword, "OUTLET" ) )
393         OUTLET[0] = (double)atof (value);
394
395     else if ( !strcmp( keyword, "VIS_WATER" ) )
396         VIS_WATER = (double)atoi( value );
397     else if ( !strcmp( keyword, "VIS_OIL" ) )
398         VIS_OIL = (double)atoi( value );
399     else if ( !strcmp( keyword, "VIS_EMUL" ) )
400         VIS_EMUL = (double)atoi( value );
401
402     else if ( !strcmp( keyword, "viscoef" ) )
403         viscoef = (double)atoi( value );
404     else if ( !strcmp( keyword, "permcoef" ) )
405         permcoef = (double)atoi( value );
406
407     else
408       MessagePopup ("Missed Data ", keyword);
409     } //While
410
411 fclose( fp );
412 return;
413 }
414
415 void SaveConfig(int type) //build based on DSN_Save_Config(int type)
416 {
417 //int i, value2;
418 //double value;
419 FILE *fp;
420
421 if(type)
422     {
423     //selecting the file to save to
424     FileSelectPopup ("", "*.cfg", "",
425                      "Enter the name of the file to save",
426                      VAL_SAVE_BUTTON, 0, 0, 1, 1, cfgfile);
427     fp = fopen( cfgfile, "w" );
428     }
429 else
430 fp = fopen( pathname, "w" );
431     //Camera
432     fprintf( fp, "Camera_Name\t%s\n", camName);
433
434 //COM PANEL
435     //COM 1 [Pump#1]
436     fprintf( fp, "P1_comport\t%i\n",        P1_comport);
437     fprintf( fp, "P1_baudrate\t%i\n",       P1_baudrate);
438     fprintf( fp, "P1_parity\t%i\n",         P1_parity);
439     fprintf( fp, "P1_databits\t%i\n",       P1_databits);
440     fprintf( fp, "P1_stopbits\t%i\n",       P1_stopbits);
441     fprintf( fp, "P1_inputq\t%i\n",         P1_inputq);
442     fprintf( fp, "P1_outputq\t%i\n",        P1_outputq);
443     fprintf( fp, "P1_ctsmode\t%i\n",        P1_ctsmode);
444     fprintf( fp, "P1_xmode\t%i\n",          P1_xmode);
445     fprintf( fp, "P1_timeout\t%f\n",        P1_timeout);
446     fprintf( fp, "P1_portindex\t%i\n",      P1_portindex);
447
448     //COM 2 [Pump#2]
449     fprintf( fp, "P2_comport\t%i\n",        P2_comport);
450     fprintf( fp, "P2_baudrate\t%i\n",       P2_baudrate);
451     fprintf( fp, "P2_parity\t%i\n",         P2_parity);
452     fprintf( fp, "P2_databits\t%i\n",       P2_databits);
453     fprintf( fp, "P2_stopbits\t%i\n",       P2_stopbits);
454     fprintf( fp, "P2_inputq\t%i\n",         P2_inputq);
455     fprintf( fp, "P2_outputq\t%i\n",        P2_outputq);
456     fprintf( fp, "P2_ctsmode\t%i\n",        P2_ctsmode);
457     fprintf( fp, "P2_xmode\t%i\n",          P2_xmode);
458     fprintf( fp, "P2_timeout\t%f\n",        P2_timeout);
459     fprintf( fp, "P2_portindex\t%i\n",      P2_portindex);
460
461     //COM 3 [Pump#3]
462     fprintf( fp, "P3_comport\t%i\n",        P3_comport);
463     fprintf( fp, "P3_baudrate\t%i\n",       P3_baudrate);
464     fprintf( fp, "P3_parity\t%i\n",         P3_parity);
465     fprintf( fp, "P3_databits\t%i\n",       P3_databits);
466     fprintf( fp, "P3_stopbits\t%i\n",       P3_stopbits);
467     fprintf( fp, "P3_inputq\t%i\n",         P3_inputq);
468     fprintf( fp, "P3_outputq\t%i\n",        P3_outputq);
469     fprintf( fp, "P3_ctsmode\t%i\n",        P3_ctsmode);
470     fprintf( fp, "P3_xmode\t%i\n",          P3_xmode);
471     fprintf( fp, "P3_timeout\t%f\n",        P3_timeout);
472     fprintf( fp, "P3_portindex\t%i\n",      P3_portindex);
473
474     // GRAPH PANEL
475     // Graph #1
476     fprintf( fp, "plotVar_G_1\t%i\n",       plotVar_G_1);
477     fprintf( fp, "X_Range_G_1\t%f\n",       X_Range_G_1);
478     fprintf( fp, "Y_Mode_G_1\t%i\n",        Y_Mode_G_1);
479     fprintf( fp, "Y_Min_G_1\t%f\n",         Y_Min_G_1);
480     fprintf( fp, "Y_Max_G_1\t%f\n",         Y_Max_G_1);
481     // Graph #2
482     fprintf( fp, "plotVar_G_2\t%i\n",       plotVar_G_2);
483     fprintf( fp, "X_Range_G_2\t%f\n",       X_Range_G_2);
484     fprintf( fp, "Y_Mode_G_2\t%i\n",        Y_Mode_G_2);
```

```
485     fprintf( fp, "Y_Min_G_2\t%f\n",         Y_Min_G_2);
486     fprintf( fp, "Y_Max_G_2\t%f\n",         Y_Max_G_2);
487     // Graph #3
488     fprintf( fp, "plotVar_G_3\t%i\n",       plotVar_G_3);
489     fprintf( fp, "X_Range_G_3\t%f\n",       X_Range_G_3);
490     fprintf( fp, "Y_Mode_G_3\t%i\n",        Y_Mode_G_3);
491     fprintf( fp, "Y_Min_G_3\t%f\n",         Y_Min_G_3);
492     fprintf( fp, "Y_Max_G_3\t%f\n",         Y_Max_G_3);
493     // Graph #4
494     fprintf( fp, "plotVar_G_4\t%i\n",       plotVar_G_4);
495     fprintf( fp, "X_Range_G_4\t%f\n",       X_Range_G_4);
496     fprintf( fp, "Y_Mode_G_4\t%i\n",        Y_Mode_G_4);
497     fprintf( fp, "Y_Min_G_4\t%f\n",         Y_Min_G_4);
498     fprintf( fp, "Y_Max_G_4\t%f\n",         Y_Max_G_4);
499
500     // Equipment Setup
501     fprintf( fp, "Exp_Type\t%i\n",          Exp_Type);
502
503     //Logging Variables
504     fprintf( fp, "Is_P1\t%i\n",             Is_P1);
505     fprintf( fp, "Is_P2\t%i\n",             Is_P2);
506     fprintf( fp, "Is_P3\t%i\n",             Is_P3);
507     fprintf( fp, "Is_M1\t%i\n",             Is_M1);
508     fprintf( fp, "Is_M2\t%i\n",             Is_M2);
509     fprintf( fp, "Is_M3\t%i\n",             Is_M3);
510     fprintf( fp, "Is_M4\t%i\n",             Is_M4);
511     fprintf( fp, "Is_PRES\t%i\n",           Is_PRES);
512     fprintf( fp, "Is_BPR\t%i\n",            Is_BPR);
513     fprintf( fp, "Is_Graph\t%i\n",          Is_Graph);
514     fprintf( fp, "Is_DAQ\t%i\n",            Is_DAQ);
515     fprintf( fp, "Is_Log\t%i\n",            Is_Log);
516
517     fprintf( fp, "ZEROTP2\t%f\n",           ZEROTP2[0]); //upstream
518     fprintf( fp, "ZERODP1\t%f\n",           ZERODP1[0]);
519     fprintf( fp, "ZEROTP1\t%f\n",           ZEROTP1[0]); //downstream
520
521     fprintf( fp, "ZEROCP11\t%f\n",          ZEROCP11[0]);
522     fprintf( fp, "ZEROCP12\t%f\n",          ZEROCP12[0]);
523     fprintf( fp, "ZEROCP13\t%f\n",          ZEROCP13[0]);
524     fprintf( fp, "ZEROCP14\t%f\n",          ZEROCP14[0]);
525     fprintf( fp, "ZEROCP15\t%f\n",          ZEROCP15[0]);
526
527     fprintf( fp, "ZEROCP21\t%f\n",          ZEROCP21[0]);
528     fprintf( fp, "ZEROCP22\t%f\n",          ZEROCP22[0]);
529     fprintf( fp, "ZEROCP23\t%f\n",          ZEROCP23[0]);
530     fprintf( fp, "ZEROCP24\t%f\n",          ZEROCP24[0]);
531     fprintf( fp, "ZEROCP25\t%f\n",          ZEROCP25[0]);
532
533     fprintf( fp, "ZEROCP31\t%f\n",          ZEROCP31[0]);
534     fprintf( fp, "ZEROCP32\t%f\n",          ZEROCP32[0]);
535     fprintf( fp, "ZEROCP33\t%f\n",          ZEROCP33[0]);
536     fprintf( fp, "ZEROCP34\t%f\n",          ZEROCP34[0]);
537     fprintf( fp, "ZEROCP35\t%f\n",          ZEROCP35[0]);
538
539
540     fprintf( fp, "G_WATERINJ\t%f\n",        G_WATERINJ[1]);
541     fprintf( fp, "G_OILINJ\t%f\n",          G_OILINJ[1]);
542     fprintf( fp, "G_EMULINJ\t%f\n",         G_EMULINJ[1]);
543
544     fprintf( fp, "OUTLET\t%f\n",            OUTLET[1]);
545
546     fprintf( fp, "VIS_WATER\t%f\n",         VIS_WATER);
547     fprintf( fp, "VIS_OIL\t%f\n",           VIS_OIL);
548     fprintf( fp, "VIS_EMUL\t%f\n",          VIS_EMUL);
549
550     fprintf( fp, "viscoef\t%f\n",           viscoef);
551     fprintf( fp, "permcoef\t%f\n",          permcoef);
552
553 fclose( fp );
554 return;
555 }
556
557 void EnableP1PanelControls (int enable)   /* PUMP #1 buttons activation */
558 {
559     //pump
560     SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_1, ATTR_DIMMED, enable);
561     SetCtrlAttribute (panel_handle, MAIN_P1_START, ATTR_DIMMED, enable);
562     SetCtrlAttribute (panel_handle, MAIN_P1_STOP, ATTR_DIMMED, enable);
563     SetCtrlAttribute (panel_handle, MAIN_P1_PURGE, ATTR_DIMMED, enable);
564     SetCtrlAttribute (panel_handle, MAIN_P1_RESTART, ATTR_DIMMED, enable);
565     SetCtrlAttribute (panel_handle, MAIN_P1_5ERR, ATTR_DIMMED, enable);
566     SetCtrlAttribute (panel_handle, MAIN_P1_SET_NFR, ATTR_DIMMED, enable);
567     SetCtrlAttribute (panel_handle, MAIN_P1_SNFR, ATTR_DIMMED, enable);
568     SetCtrlAttribute (panel_handle, MAIN_STATUS_P_1, ATTR_DIMMED, enable);
569     SetCtrlAttribute (panel_handle, MAIN_P1_TEXTMSG, ATTR_DIMMED, enable);
570 }
571
572 void EnableMassTurnControls (int enable) /* Allow user Connect|Disconnect with Mass flow meters */
573 {
574     SetCtrlAttribute (config_handle, CONFIG_CON_M_1, ATTR_DIMMED, enable);
575     SetCtrlAttribute (config_handle, CONFIG_CON_M_2, ATTR_DIMMED, enable);
576     SetCtrlAttribute (config_handle, CONFIG_CON_M_3, ATTR_DIMMED, enable);
577     SetCtrlAttribute (config_handle, CONFIG_CON_M_4, ATTR_DIMMED, enable);
578     SetCtrlAttribute (config_handle, CONFIG_CON_BPR, ATTR_DIMMED, enable);
579 }
580
581 void EnableM1PanelControls (int enable)   /* MASS FLOW METER #1 buttons activation */
```

```
{
    //mass flow meter
    SetCtrlAttribute (panel_handle, MAIN_DENSITY_M_1, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_TEMP_M_1, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_MFR_M_1, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA1_M_1, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA2_M_1, ATTR_DIMMED, enable);
}

void EnableM2PanelControls (int enable)   /* MASS FLOW METER #2 buttons activation */
{
    //mass flow meter
    SetCtrlAttribute (panel_handle, MAIN_DENSITY_M_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_TEMP_M_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_MFR_M_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA1_M_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA2_M_2, ATTR_DIMMED, enable);
}

void EnableM3PanelControls (int enable)   /* MASS FLOW METER #3 buttons activation */
{
    //mass flow meter
    SetCtrlAttribute (panel_handle, MAIN_DENSITY_M_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_TEMP_M_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_MFR_M_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA1_M_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA2_M_3, ATTR_DIMMED, enable);
}

void EnableM4PanelControls (int enable)   /* MASS FLOW METER #4 buttons activation */
{
    //mass flow meter
    SetCtrlAttribute (panel_handle, MAIN_DENSITY_M_4, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_TEMP_M_4, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_MFR_M_4, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA1_M_4, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_EXTRA2_M_4, ATTR_DIMMED, enable);
}

void EnableBPRPanelControls (int enable) /* BACK PRESSURE REGULATOR buttons activation */
{
    //back pressure regulator
    SetCtrlAttribute (panel_handle, MAIN_BPR1, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_BPR_SET, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_BPR_ALARM, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_BPR_SETPOINT, ATTR_DIMMED, enable);
}

/*---------------------------------------------------------------------------*/
/* close the configuration panel.                                            */
/*---------------------------------------------------------------------------*/
int CVICALLBACK CloseConfigCallback (int panel, int control, int event,
                                     void *callbackData, int eventData1,
                                     int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT :
            HidePanel (config_handle);
             break;
        case EVENT_RIGHT_CLICK:
             break;
        }
    return(0);
}

//*********************************************************************//
//*********************************************************************//
//*********************************************************************//
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////PPPP  U   U M   M PPPP   SSS/////////////////////
////////////////////P   P U   U MM MM P   P S   /////////////////////
////////////////////PPPP  U   U M M M PPPP   SSS/////////////////////
////////////////////P     U   U M   M P         S/////////////////////
////////////////////P      UUU  M   M P     SSSS /////////////////////
//////////////////////////////com/////////////////////////////////////
////////////////////////////configuration/////////////////////////////
//*********************************************************************//
//*********************************************************************//
//*********************************************************************//

                // COM#1 //

                int CVICALLBACK Pump1ConfigCallback (int panel, int control, int event,
                        void *callbackData, int eventData1, int eventData2)
                {
                    switch (event)
                    {
                        case EVENT_COMMIT:      //sitting at COM2  (inside the program, read as COM1)

                            GetCtrlVal(config_handle, CONFIG_P1, &p1check); //get position of rig.

                            if (p1check == 1) COM1_OPEN();    // open/close connection
                            else
                            {
                                if (P1 port open)
```

```
679                                 {
680                                         outqlen = GetOutQLen (P1_comport);
681                                         if (outqlen > 0)
682                                         {
683                                             MessagePopup ("RS232 Message", "The output queue has\n"
684                                                           "data in it. Wait for device to receive\n"
685                                                           "the data or flush the queue.\n");
686                                         break;
687                                         }
688                                         RS232Error = CloseCom (P1_comport);
689                                         if (RS232Error)
690                                           DisplayRS232Error ();
691                                         SetCtrlVal (config_handle,  CONFIG_P1_LED,   0);
692                                         SetCtrlVal (panel_handle,  MAIN_P1_LED,   0);
693                                         SetCtrlAttribute (panel_handle, MAIN_WATER, ATTR_DIMMED, 1); //turn off dimmed status
694
695                                 P1_port_open=0;
696                                 }
697                                 else  MessagePopup ("COM1 Port Message", "There was no open connection.");
698                             }
699                             break;
700                         case EVENT_RIGHT_CLICK:
701                             break;
702                     }
703                     return 0;
704                 }
705
706                     //------------------------------------------------------------
707                     // COM1_OPEN        : Opens the com #1 port
708                     //------------------------------------------------------------
709
710                     void COM1_OPEN(void)
711                     { int test;
712
713
714                     P1_GetConfig();
715
716                     P1_port_open = 1;   /* initialize flag to 0 - unopened */
717                     GetConfigParms ();
718                     DisableBreakOnLibraryErrors ();
719                     RS232Error = OpenComConfig (P1_comport, "", P1_baudrate, P1_parity,
720                                   P1_databits, P1_stopbits, P1_inputq, P1_outputq);
721                     SetCtrlVal  (config_handle, CONFIG_NUM, RS232Error);  // show up RS232 error
722                     EnableBreakOnLibraryErrors ();
723
724                     if (RS232Error) DisplayRS232Error ();
725                     if (RS232Error == 0)
726                                     {
727                                     P1_port_open = 1;
728                                     SetCtrlVal  (config_handle, CONFIG_NUM, P1_port_open);
729
730                                     GetCtrlVal (config_handle, CONFIG_P1_XMODE, &P1_xmode);
731                                     SetXMode (P1_comport, P1_xmode);
732
733                                     GetCtrlVal (config_handle, CONFIG_P1_CTSMODE, &P1_ctsmode);
734                                     SetCTSMode (P1_comport, P1_ctsmode);
735
736                                     GetCtrlVal (config_handle, CONFIG_P1_TIMEOUT, &P1_timeout);
737                                     SetComTime (P1_comport, P1_timeout);
738
739
740                                     }
741
742                     if (P1_port_open)
743                         {
744                         SetCtrlAttribute (config_handle,  CONFIG_P1_LED,   ATTR_DIMMED, 0);
745
746                         test=1;   // show that COM #1 was opened
747                         if(test)
748                             {
749                             SetCtrlVal (config_handle,  CONFIG_P1_LED,   1);
750                             SetCtrlVal (panel_handle,  MAIN_P1_LED,   1);
751                             SetCtrlAttribute (panel_handle, MAIN_WATER, ATTR_DIMMED, 0);
752                             }
753                         else
754                             {
755                             SetCtrlVal (config_handle,  CONFIG_P1_LED,   0);
756                             SetCtrlVal (panel_handle,  MAIN_P1_LED,   0);
757                             SetCtrlAttribute (panel_handle, MAIN_WATER, ATTR_DIMMED, 1);
758                             }
759                         }
760                     return;
761                     }
762
763                                 void P1_GetConfig (void)
764                                 {
765                                     GetConfigParms();
766                                         #ifdef _NI_unix_
767                                             P1_devicename[0]=0;
768                                         #else
769                                             GetLabelFromIndex (config_handle, CONFIG_P1_COMPORT, P1_portindex,
770                                                                P1_devicename);
771                                         #endif
772                                 }
773
774                 // COM#2 //
775
```

```
776  int CVICALLBACK Pump2ConfigCallback (int panel, int control, int event,
777          void *callbackData, int eventData1, int eventData2)
778  {
779      switch (event)
780      {
781          case EVENT_COMMIT:      //sitting at COM3  (inside the program, read as COM2)
782              GetCtrlVal(config_handle, CONFIG_P2, &p2check);
783              if (p2check == 1) COM2_OPEN();
784              else
785              {
786                  if (P2_port_open)
787                  {
788                          outqlen = GetOutQLen (P2_comport);
789                          if (outqlen > 0)
790                          {
791                              MessagePopup ("RS232 Message", "The output queue has\n"
792                                            "data in it. Wait for device to receive\n"
793                                            "the data or flush the queue.\n");
794                          break;
795                          }
796                          RS232Error = CloseCom (P2_comport);
797                          if (RS232Error)
798                            DisplayRS232Error ();
799                          SetCtrlVal (config_handle,  CONFIG_P2_LED,   0);
800                          SetCtrlVal (panel_handle,  MAIN_P2_LED,    0);
801                          SetCtrlAttribute (panel_handle, MAIN_OIL, ATTR_DIMMED, 1);
802
803                  P2_port_open=0;
804                  }
805                  else  MessagePopup ("COM2 Port Message", "There was no open connection.");
806              }
807              break;
808          case EVENT_RIGHT_CLICK:
809              break;
810      }
811      return 0;
812  }
813
814
815      //-----------------------------------------------------------
816      // COM2_OPEN        : Opens the com #2 port
817      //-----------------------------------------------------------
818
819      void COM2_OPEN(void)
820      { int test;
821
822
823      P2_GetConfig();
824
825      P2_port_open = 1;   /* initialize flag to 0 - unopened */
826      GetConfigParms ();
827      DisableBreakOnLibraryErrors ();
828      RS232Error = OpenComConfig (P2_comport, "", P2_baudrate, P2_parity,
829                          P2_databits, P2_stopbits, P2_inputq, P2_outputq);
830      SetCtrlVal  (config_handle, CONFIG_NUM, RS232Error);  // show up RS232 error
831      EnableBreakOnLibraryErrors ();
832
833      if (RS232Error) DisplayRS232Error ();
834      if (RS232Error == 0)
835                      {
836                      P2_port_open = 1;
837                      SetCtrlVal  (config_handle, CONFIG_NUM, P2_port_open);
838
839                      GetCtrlVal (config_handle, CONFIG_P2_XMODE, &P2_xmode);
840                      SetXMode (P2_comport, P2_xmode);
841
842                      GetCtrlVal (config_handle, CONFIG_P2_CTSMODE, &P2_ctsmode);
843                      SetCTSMode (P2_comport, P2_ctsmode);
844
845                      GetCtrlVal (config_handle, CONFIG_P2_TIMEOUT, &P2_timeout);
846                      SetComTime (P2_comport, P2_timeout);
847
848
849                      }
850
851      if (P2_port_open)
852          {
853          SetCtrlAttribute (config_handle,  CONFIG_P2_LED,   ATTR_DIMMED, 0);
854
855          test=1;   // show that COM #2 was opened
856          if(test)
857              {
858              SetCtrlVal (config_handle,  CONFIG_P2_LED,   1);
859              SetCtrlVal (panel_handle,  MAIN_P2_LED,   1);
860              SetCtrlAttribute (panel_handle, MAIN_OIL, ATTR_DIMMED, 0);
861              }
862          else
863              {
864              SetCtrlVal (config_handle,  CONFIG_P2_LED,   0);
865              SetCtrlVal (panel_handle,  MAIN_P2_LED,   0);
866              SetCtrlAttribute (panel_handle, MAIN_OIL, ATTR_DIMMED, 1);
867              }
868          }
869      return;
870      }
871
872                  void P2 GetConfig (void)
```

```
                                {
                                    GetConfigParms();

                                        #ifdef _NI_unix_
                                            P2_devicename[0]=0;
                                        #else
                                            GetLabelFromIndex (config_handle, CONFIG_P2_COMPORT, P2_portindex,
                                                               P2_devicename);
                                        #endif
                                }

            // COM#3 //

            int CVICALLBACK Pump3ConfigCallback (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
            {
                switch (event)
                {
                    case EVENT_COMMIT:        //sitting at COM5   (inside the program, read as COM3)
                        GetCtrlVal(config_handle, CONFIG_P3, &p3check);
                        if (p3check == 1) COM3_OPEN();
                        else
                        {
                            if (P3_port_open)
                            {
                                    outqlen = GetOutQLen (P3_comport);
                                    if (outqlen > 0)
                                    {
                                        MessagePopup ("RS232 Message", "The output queue has\n"
                                                      "data in it. Wait for device to receive\n"
                                                      "the data or flush the queue.\n");
                                    break;
                                    }
                                    RS232Error = CloseCom (P3_comport);
                                    if (RS232Error)
                                      DisplayRS232Error ();
                                    SetCtrlVal (config_handle,  CONFIG_P3_LED,   0);
                                    SetCtrlVal (panel_handle,  MAIN_P3_LED,    0);
                                    SetCtrlAttribute (panel_handle, MAIN_EMULSION, ATTR_DIMMED, 1);
                                //  SetCtrlVal (panel_handle, MAIN_LOG_P3, 1);
                            P3_port_open=0;
                            }
                            else  MessagePopup ("COM3 Port Message", "There was no open connection.");
                        }
                        break;
                    case EVENT_RIGHT_CLICK:
                        break;
                }
                return 0;
            }

                //-----------------------------------------------------------
                // COM3_OPEN        : Opens the com #3 port
                //-----------------------------------------------------------

                void COM3_OPEN(void)
                { int test;


                P3_GetConfig();

                P3_port_open = 1;   /* initialize flag to 0 - unopened */
                GetConfigParms ();
                DisableBreakOnLibraryErrors ();
                RS232Error = OpenComConfig (P3_comport, "", P3_baudrate, P3_parity,
                                    P3_databits, P3_stopbits, P3_inputq, P3_outputq);
                SetCtrlVal  (config_handle, CONFIG_NUM, RS232Error);  // show up RS232 error
                EnableBreakOnLibraryErrors ();

                if (RS232Error) DisplayRS232Error ();
                if (RS232Error == 0)
                                {
                                P3_port_open = 1;
                                SetCtrlVal  (config_handle, CONFIG_NUM, P3_port_open);

                                GetCtrlVal (config_handle, CONFIG_P3_XMODE, &P3_xmode);
                                SetXMode (P3_comport, P3_xmode);

                                GetCtrlVal (config_handle, CONFIG_P3_CTSMODE, &P3_ctsmode);
                                SetCTSMode (P3_comport, P3_ctsmode);

                                GetCtrlVal (config_handle, CONFIG_P3_TIMEOUT, &P3_timeout);
                                SetComTime (P3_comport, P3_timeout);


                                }

                if (P3_port_open)
                    {
                    SetCtrlAttribute (config_handle,  CONFIG_P3_LED,   ATTR_DIMMED, 0);

                    test=1;   // show that COM #3 was opened
                    if(test)
                        {
                        SetCtrlVal (config_handle,  CONFIG_P3_LED,   1);
                        SetCtrlVal (panel_handle,  MAIN_P3_LED,   1);
                        SetCtrlAttribute (panel_handle, MAIN_EMULSION, ATTR_DIMMED, 0);
```

```
970
971                             }
972                         else
973                             {
974                             SetCtrlVal (config_handle,  CONFIG_P3_LED,   0);
975                             SetCtrlVal (panel_handle,  MAIN_P3_LED,   0);
976                             SetCtrlAttribute (panel_handle, MAIN_EMULSION, ATTR_DIMMED, 1);
977                             }
978                         }
979                     return;
980                     }
981
982                                 void P3_GetConfig (void)
983                                 {
984                                     GetConfigParms();
985
986                                         #ifdef _NI_unix_
987                                             P3_devicename[0]=0;
988                                         #else
989                                             GetLabelFromIndex (config_handle, CONFIG_P3_COMPORT, P3_portindex,
990                                                                P3_devicename);
991                                         #endif
992                                 }
993
994 //*********************************************************************//
995 //*********************************************************************//
996
997 /*-------------------------------------------------------------------------*/
998 /*              Function send-read command to the pump                     */
999 /*-------------------------------------------------------------------------*/
1000
1001 void SendReadPump (void) //function valid for any pump, required comport number.
1002  {
1003
1004     //CP= current comport
1005
1006     FlushOutQ (CP);
1007     FlushInQ (CP);
1008
1009     stringsize = StringLength (send_data);
1010     bytes_sent = ComWrt (CP, send_data, stringsize);
1011     Delay(0.05); /* 0.05  */
1012
1013             read_cnt = GetInQLen (CP);
1014
1015             read_data[0] = '\0';
1016             read_term = 13;
1017             bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1018
1019             CopyString (SRP_read_data, 0, read_data, 0, bytes_read);
1020             SetCtrlVal (panel_name, PTEXT_panel_name, SRP_read_data);
1021
1022             RS232Error = ReturnRS232Err ();
1023             if (RS232Error)
1024                 DisplayRS232Error ();
1025
1026             send_data[0] = '\0';
1027
1028 }
1029
1030 void CheckPressurePump1 (void) // part of PumpsThread function. helps update information regarding pressure
1031 {
1032 int i;
1033 double probe_temp, probe_pressure, probe_pressure_psi;
1034 char temp [NUM];
1035 ////////////////////
1036 panel_name=panel_handle;
1037 send_data[0] = '\0';
1038 Fmt (send_data, "%s<PRESSURE?\r");
1039 CP=P1_comport;
1040 text_panel_name= MAIN_PRESSURE_P_1;
1041 ////////////////////
1042
1043         FlushOutQ (CP);
1044         FlushInQ (CP);
1045
1046         stringsize = StringLength (send_data);
1047         bytes_sent = ComWrt (CP, send_data, stringsize);
1048         Delay(0.05); /* 0.05  */
1049
1050                 read_cnt = GetInQLen (CP);
1051
1052                 read_data[0] = '\0';
1053                 read_term = 13;
1054                 bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1055
1056                 CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
1057
1058                 for (i=9; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
1059                     temp[i-9]=tbox_read_data[i];
1060                     temp[i-9]='\0';         //close by '\0'
1061
1062                 probe_pressure = atof(temp) /10 ;
1063                 probe_pressure_psi=probe_pressure * 14.5037738;
1064     ProcessSystemEvents ();
1065                 SetCtrlVal (panel_name, text_panel_name, probe_pressure_psi);
1066
```

```
1067
1068     CmtGetLock (Pumps_lockHandle);
1069     ////////////////////
1070     DSN_SHIFT_Slow(3); //pump #1
1071
1072     P11[0] = probe_pressure_psi;            // Log value in P11 Array
1073     ////////////////////
1074     CmtReleaseLock (Pumps_lockHandle);
1075
1076                 RS232Error = ReturnRS232Err ();
1077                 if (RS232Error)
1078                     DisplayRS232Error ();
1079
1080                 send_data[0] = '\0';
1081
1082 }
1083
1084 void CheckPressurePump2 (void) // part of PumpsThread function. helps update information regarding pressure
1085 {
1086 int i;
1087 double probe_temp, probe_pressure, probe_pressure_psi;
1088 char temp [NUM];
1089 ////////////////////
1090 panel_name=panel_handle;
1091 send_data[0] = '\0';
1092 Fmt (send_data, "%s<PRESSURE?\r");
1093 CP=P2_comport;
1094 text_panel_name= MAIN_PRESSURE_P_2;
1095 ////////////////////
1096
1097         FlushOutQ (CP);
1098         FlushInQ (CP);
1099
1100         stringsize = StringLength (send_data);
1101         bytes_sent = ComWrt (CP, send_data, stringsize);
1102         Delay(0.05); /* 0.05  */
1103
1104                 read_cnt = GetInQLen (CP);
1105
1106                 read_data[0] = '\0';
1107                 read_term = 13;
1108                 bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1109
1110                 CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
1111
1112                 for (i=9; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
1113                     temp[i-9]=tbox_read_data[i];
1114                     temp[i-9]='\0';         //close by '\0'
1115
1116                 probe_pressure = atof(temp)/10;
1117                 probe_pressure_psi=probe_pressure * 14.5037738;
1118     ProcessSystemEvents ();
1119                 SetCtrlVal (panel_name, text_panel_name, probe_pressure_psi);
1120
1121
1122     CmtGetLock (Pumps_lockHandle);
1123     ////////////////////
1124     DSN_SHIFT_Slow(4); //pump #2
1125
1126     P21[0] = probe_pressure_psi;            // Log value in P21 Array
1127     ////////////////////
1128     CmtReleaseLock (Pumps_lockHandle);
1129
1130                 RS232Error = ReturnRS232Err ();
1131                 if (RS232Error)
1132                     DisplayRS232Error ();
1133
1134                 send_data[0] = '\0';
1135
1136 }
1137
1138 void CheckPressurePump3 (void) // part of PumpsThread function. helps update information regarding pressure
1139 {
1140 int i;
1141 double probe_temp, probe_pressure, probe_pressure_psi;
1142 char temp [NUM];
1143 ////////////////////
1144 panel_name=panel_handle;
1145 send_data[0] = '\0';
1146 Fmt (send_data, "%s<PRESSURE?\r");
1147 CP=P3_comport;
1148 text_panel_name= MAIN_PRESSURE_P_3;
1149 ////////////////////
1150
1151
1152
1153         FlushOutQ (CP);
1154         FlushInQ (CP);
1155
1156         stringsize = StringLength (send_data);
1157         bytes_sent = ComWrt (CP, send_data, stringsize);
1158         Delay(0.05); /* 0.05  */
1159
1160                 read_cnt = GetInQLen (CP);
1161
1162                 read_data[0] = '\0';
1163                 read_term = 13;
```

```
1164                bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1165
1166                CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
1167
1168
1169                for (i=9; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
1170                    temp[i-9]=tbox_read_data[i];
1171                    temp[i-9]='\0';         //close by '\0'
1172
1173                probe_pressure = atof(temp)/10;
1174                probe_pressure_psi=probe_pressure * 14.5037738;
1175    ProcessSystemEvents ();
1176                SetCtrlVal (panel_name, text_panel_name, probe_pressure_psi);
1177
1178
1179    CmtGetLock (Pumps_lockHandle);
1180    ////////////////////
1181    DSN_SHIFT_Slow(5); //pump #2
1182    P31[0] = probe_pressure_psi;            // Log value in P21 Array
1183    ////////////////////
1184    CmtReleaseLock (Pumps_lockHandle);
1185
1186
1187
1188                RS232Error = ReturnRS232Err ();
1189                if (RS232Error)
1190                    DisplayRS232Error ();
1191
1192                send_data[0] = '\0';
1193 }
1194
1195 /*---------------------------------------------------------------------------*/
1196 /*              Function SET NEW FLOW RATE command to the pump               */
1197 /*---------------------------------------------------------------------------*/
1198
1199 void SetPumpFlowrate(void) /* build based on DSN_Set_Bath_Setpoint*/
1200 {
1201         //CP= current comport
1202         //LD= LED to set ON
1203
1204         char  ascii_setpoint[15],
1205               wholeNum_string[4],
1206               decNum_string[4];
1207
1208         char* setpointcode = "F";
1209
1210
1211
1212         int wholeNum,
1213             decNum;
1214
1215         wholeNum = wb_setpoint;                     // Captures the whole number portion of the setpoint
1216         decNum = (wb_setpoint-wholeNum)*1000;       // Captures the 2 decimal places of the setpoint
1217
1218         sprintf(wholeNum_string, "%i", wholeNum);
1219         sprintf(decNum_string, "%i", decNum);
1220
1221 if (decNum == 0)
1222         {
1223         // Assembles the command as shown below             e.g.
1224         strcpy(ascii_setpoint, setpointcode);       // F     F
1225         strcat(ascii_setpoint, wholeNum_string);    // FX    F1
1226         strcat(ascii_setpoint, decNum_string);      // FXX   F10
1227         strcat(ascii_setpoint, decNum_string);      // FXXX  F100
1228         strcat(ascii_setpoint, decNum_string);      // FXXXX F1000
1229         strcat(ascii_setpoint, "\r");               // FXXXX\r
1230         }
1231         else
1232         {
1233             // Assembles the command as shown below
1234             strcpy(ascii_setpoint, setpointcode);       // F
1235             strcat(ascii_setpoint, wholeNum_string);    // FXXX
1236             strcat(ascii_setpoint, decNum_string);      // FXXXXX
1237             strcat(ascii_setpoint, "\r");               // FXXXXX\r
1238         }
1239
1240         FlushOutQ (CP);
1241         FlushInQ (CP);
1242
1243         stringsize = StringLength (ascii_setpoint);
1244         ComWrt (CP, ascii_setpoint, stringsize);    // Sends new setpoint to COM Port
1245
1246
1247         Delay(0.5);
1248
1249         read_cnt = GetInQLen (CP);
1250
1251         read_term = 13;
1252
1253         bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1254         CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
1255
1256
1257
1258     /*  SetCtrlVal (panel_handle, SERIAL_FLOWRATE_R, tbox_read_data);   */
1259
1260
```

```
1261         if (tbox_read_data && "OK")
1262                SetCtrlVal (panel_handle, LED, 1);
1263         else
1264         {
1265                SetCtrlVal (panel_handle, LED, 0);
1266                MessagePopup ("Error","flow rate has not changed");
1267         }
1268
1269         RS232Error = ReturnRS232Err ();
1270             if (RS232Error)
1271                 DisplayRS232Error ();
1272         /* Return LED back to the initial value */
1273         Delay(0.5);
1274         SetCtrlVal (panel_handle, LED, 0);
1275 }
1276
1277 // Pump #1 Start/Stop/Reset/5errors/Set new flow rate
1278
1279 int CVICALLBACK P1SFCallBack (int panel, int control, int event,
1280         void *callbackData, int eventData1, int eventData2)
1281 {
1282     switch (event)
1283     {
1284         case EVENT_COMMIT:
1285
1286 CmtGetLock (Pumps_lockHandle);
1287
1288                Delay (0.1);
1289                panel_name=panel_handle;
1290                CP=P1_comport;
1291                send_data[0] = '\0';
1292                Fmt (send_data, "%s<ON\r");
1293                PTEXT_panel_name= MAIN_STATUS_P_1; //show response from pump
1294                SendReadPump ();
1295
1296                pumprun_1 = 11;
1297
1298 CmtReleaseLock (Pumps_lockHandle);
1299                break;
1300         case EVENT_RIGHT_CLICK :
1301             MessagePopup ("Pump #1", "Will run Pump at current flow rate.");
1302             break;
1303     }
1304     return 0;
1305 }
1306
1307 int CVICALLBACK P1STOPCallBack (int panel, int control, int event,
1308         void *callbackData, int eventData1, int eventData2)
1309 {
1310     switch (event)
1311     {
1312         case EVENT_COMMIT:
1313 CmtGetLock (Pumps_lockHandle);
1314                Delay (0.1);
1315                panel_name=panel_handle;
1316                CP=P1_comport;
1317                send_data[0] = '\0';
1318                Fmt (send_data, "%s<OFF\r");
1319                PTEXT_panel_name= MAIN_STATUS_P_1;
1320                SendReadPump ();
1321
1322                pumprun_1 = 12;
1323
1324 CmtReleaseLock (Pumps_lockHandle);
1325             break;
1326     }
1327     return 0;
1328 }
1329
1330 int CVICALLBACK P1PURGECallBack (int panel, int control, int event,
1331         void *callbackData, int eventData1, int eventData2)
1332 {
1333     switch (event)
1334     {
1335         case EVENT_COMMIT:
1336                panel_name=panel_handle;
1337                CP=P1_comport;
1338                send_data[0] = '\0';
1339                Fmt (send_data, "%s<PURGE\r");
1340                PTEXT_panel_name= MAIN_STATUS_P_1;
1341                SendReadPump ();
1342             break;
1343     }
1344     return 0;
1345 }
1346
1347 int CVICALLBACK P1RESTARTCallBack (int panel, int control, int event,
1348         void *callbackData, int eventData1, int eventData2)
1349 {
1350     switch (event)
1351     {
1352         case EVENT_COMMIT:
1353
1354                panel_name=panel_handle;
1355                CP=P1_comport;
1356                send_data[0] = '\0';
1357                Fmt (send data, "%s<RESET\r");
```

```
1358                PTEXT_panel_name= MAIN_STATUS_P_1;
1359                SendReadPump ();
1360
1361            break;
1362    }
1363    return 0;
1364 }
1365
1366 int CVICALLBACK P15ERRCallBack (int panel, int control, int event,
1367         void *callbackData, int eventData1, int eventData2)
1368 {
1369    switch (event)
1370    {
1371        case EVENT_COMMIT:
1372
1373                panel_name=panel_handle;
1374                CP=P1_comport;
1375                send_data[0] = '\0';
1376                Fmt (send_data, "%s<ERRORS?\r");
1377                PTEXT_panel_name= MAIN_STATUS_P_1;
1378                SendReadPump ();
1379
1380            break;
1381    }
1382    return 0;
1383 }
1384
1385 int CVICALLBACK P1SNFRCallBack (int panel, int control, int event,
1386         void *callbackData, int eventData1, int eventData2)
1387 {
1388 switch (event)
1389    {
1390        case EVENT_COMMIT:
1391            CmtGetLock (Pumps_lockHandle);
1392            GetCtrlVal(panel_handle, MAIN_P1_SET_NFR, &wb_setpoint);
1393            SNFRP_1();
1394            CmtReleaseLock (Pumps_lockHandle);
1395            break;
1396    }
1397    return 0;
1398 }
1399
1400 void SNFRP_1 (void)
1401 {
1402 int i;
1403 double probe_temp;
1404 char temp [NUM];
1405
1406                CP=P1_comport;
1407                LED=MAIN_P1_OK_LED;
1408
1409                SetPumpFlowrate();
1410
1411                Delay(0.2);
1412                //update panel with new flow rate value//
1413
1414                     ////////////
1415                     panel_name=panel_handle;
1416                     CP=P1_comport;
1417                     text_panel_name= MAIN_WAS_FR_P_1;
1418                     ////////////
1419
1420                     FlushOutQ (CP);
1421                     FlushInQ (CP);
1422
1423                     send_data[0] = '\0';
1424                     Fmt (send_data, "%s<F?\r");
1425                     stringsize = StringLength (send_data);
1426
1427                     bytes_sent = ComWrt (CP, send_data, stringsize);
1428                     Delay(0.05); /* 0.05  */
1429
1430                     read_cnt = GetInQLen (CP);
1431
1432                     read_data[0] = '\0';
1433                     read_term = 13;
1434
1435                     bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
1436
1437                     CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
1438
1439                         for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
1440                             temp[i-2]=tbox_read_data[i];
1441                             temp[i-2]='\0';          //close by '\0'
1442
1443                     probe_temp = atof(temp);
1444                     ProcessSystemEvents ();
1445                     SetCtrlVal (panel_name, text_panel_name, probe_temp);
1446
1447                     PFR11[0] = probe_temp;      // Log value in PFR11 Array
1448
1449                     RS232Error = ReturnRS232Err ();
1450                     if (RS232Error)
1451                         DisplayRS232Error ();
1452
1453                     send_data[0] = '\0';
1454
```

```
1455
1456
1457 return;
1458 }
1459
1460 // Pump #1 Start/Stop/Reset/5errors/Set new flow rate
1461
1462 /*---------------------------------------------------------------------------*/
1463 /* Display pertinent error information.                                      */
1464 /*---------------------------------------------------------------------------*/
1465 int CVICALLBACK ErrorCallBack (int panel, int control, int event,
1466                                void *callbackData, int eventData1,
1467                                int eventData2)
1468 {
1469     switch (event)
1470         {
1471         case EVENT_COMMIT:
1472                 RS232Error = ReturnRS232Err ();
1473                 DisplayRS232Error ();
1474             break;
1475         case EVENT_RIGHT_CLICK :
1476             break;
1477         }
1478     return 0;
1479 }
1480
1481 /*---------------------------------------------------------------------------*/
1482 /* Get the status of the COM port and display it to the user.                */
1483 /*---------------------------------------------------------------------------*/
1484 int CVICALLBACK ComStatusCallBack (int panel, int control, int event,
1485                                    void *callbackData, int eventData1,
1486                                    int eventData2)
1487 {
1488     if (event == EVENT_COMMIT)
1489         {
1490             GetCtrlVal(config_handle, CONFIG_COMNUMBER, &com_num);
1491
1492             //help to protect from crashing, if some of com-ports was not active
1493             GetCtrlVal(config_handle, CONFIG_P1, &p1check);
1494             GetCtrlVal(config_handle, CONFIG_P2, &p2check);
1495             GetCtrlVal(config_handle, CONFIG_P3, &p3check);
1496
1497             if (com_num == 0)
1498             MessagePopup ("COM-service Message", "Select COM-port from the list");
1499             else if (com_num == 1 && p1check == 1)
1500             {
1501                 com_status = GetComStat (P1_comport);
1502                 DisplayComStatus ();
1503             }
1504
1505                 else if (com_num == 2 && p2check == 1)
1506                 {
1507                     com_status = GetComStat (P2_comport);
1508                     DisplayComStatus ();
1509                 }
1510                 else if (com_num == 2 && p2check == 0)
1511                 MessagePopup ("COM-service Message", "COM port activation is required");
1512
1513
1514                     else if (com_num == 3 && p3check == 1)
1515                     {
1516                         com_status = GetComStat (P3_comport);
1517                         DisplayComStatus ();
1518                     }
1519                     else if (com_num == 3 && p2check == 0)
1520                     MessagePopup ("COM-service Message", "COM port activation is required");
1521
1522         }
1523     return 0;
1524 }
1525
1526 /*---------------------------------------------------------------------------*/
1527 /* Display status information to the user.                                   */
1528 /*---------------------------------------------------------------------------*/
1529 void DisplayComStatus ()
1530 {
1531     com_msg[0] = '\0';
1532     if (com_status & 0x0001)
1533         strcat (com_msg, "Input lost: Input queue"
1534                 " filled and characters were lost.\n");
1535     if (com_status & 0x0002)
1536         strcat (com_msg, "Asynch error: Problem "
1537                 "determining number of characters in input queue.\n");
1538     if (com_status & 0x0010)
1539         strcat (com_msg, "Parity error.\n");
1540     if (com_status & 0x0020)
1541         strcat (com_msg, "Overrun error: Received"
1542                 " characters were lost.\n");
1543     if (com_status & 0x0040)
1544         strcat (com_msg, "Framing error: Stop bits were not received"
1545                 " as expected.\n");
1546     if (com_status & 0x0080)
1547         strcat (com_msg, "Break: A break signal was detected.\n");
1548     if (com_status & 0x1000)
1549         strcat (com_msg, "Remote XOFF: An XOFF character was received."
1550                 "\nIf XON/XOFF was enabled, no characters are removed"
1551                 " from the output queue and sent to another device "
```

```
1552                "until that device sends an XON character.\n");
1553    if (com_status & 0x2000)
1554        strcat (com_msg, "Remote XON: An XON character was received."
1555                "\nTransmisson can resume.\n");
1556    if (com_status & 0x4000)
1557        strcat (com_msg, "Local XOFF: An XOFF character was sent to\n"
1558                " the other device.  If XON/XOFF was enabled, XOFF is\n"
1559                " transmitted when the input queue is 50%, 75%, and 90%\n"
1560                " full.\n");
1561    if (com_status & 0x8000)
1562        strcat (com_msg, "Local XON: An XON character was sent to\n"
1563                " the other device.  If XON/XOFF was enabled, XON is\n"
1564                " transmitted when the input queue empties after XOFF\n"
1565                " was sent.  XON tells the other device that it can \n"
1566                " resume sending data.\n");
1567    if (strlen (com_msg) == 0)
1568        strcat (com_msg, "No status bits are set.");
1569    MessagePopup ("RS232 Message", com_msg);
1570 }
1571
1572 /*---------------------------------------------------------------------------*/
1573 /* Display Help on the Input Queue control.                                  */
1574 /*---------------------------------------------------------------------------*/
1575 int CVICALLBACK InputQCallBack (int panel, int control, int event,
1576                                 void *callbackData, int eventData1,
1577                                 int eventData2)
1578 {
1579     if (event == EVENT_RIGHT_CLICK)
1580         DisplayHelp (InputqHelp);
1581     return 0;
1582 }
1583
1584 /*---------------------------------------------------------------------------*/
1585 /* Check the status of the queues and then terminate the RunUserInterface    */
1586 /* loop and end the application...or just provide help.                      */
1587 /*---------------------------------------------------------------------------*/
1588 int CVICALLBACK QuitCallBack (int panel, int control, int event,
1589                               void *callbackData, int eventData1,
1590                               int eventData2)
1591 {
1592     int i,test2;
1593
1594     switch (event)
1595         {
1596         case EVENT_COMMIT:
1597
1598             SaveVars();
1599             SaveConfig(0);
1600
1601
1602         /*   Add switching solenoids into initial position
1603             AV1_ONOFF
1604             AV2_ONOFF
1605             AV3_ONOFF
1606                       */
1607
1608             if (P1_port_open)
1609                 {
1610                 outqlen = GetOutQLen (P1_comport);
1611                 if (outqlen > 0)
1612                     {
1613                     MessagePopup ("RS232 Message", "The output queue has\n"
1614                                   "data in it. Wait for device to receive\n"
1615                                   "the data or flush the queue.\n");
1616                     break;
1617                     }
1618                 RS232Error = CloseCom (P1_comport);
1619                 if (RS232Error)
1620                     DisplayRS232Error ();
1621                 }
1622
1623              if (P2_port_open)
1624                 {
1625                 outqlen = GetOutQLen (P2_comport);
1626                 if (outqlen > 0)
1627                     {
1628                     MessagePopup ("RS232 Message", "The output queue has\n"
1629                                   "data in it. Wait for device to receive\n"
1630                                   "the data or flush the queue.\n");
1631                     break;
1632                     }
1633                 RS232Error = CloseCom (P2_comport);
1634                 if (RS232Error)
1635                     DisplayRS232Error ();
1636                 }
1637
1638
1639               if (P3_port_open)
1640                 {
1641                 outqlen = GetOutQLen (P3_comport);
1642                 if (outqlen > 0)
1643                     {
1644                     MessagePopup ("RS232 Message", "The output queue has\n"
1645                                   "data in it. Wait for device to receive\n"
1646                                   "the data or flush the queue.\n");
1647                     break;
1648                     }
```

```
1649                RS232Error = CloseCom (P3_comport);
1650                if (RS232Error)
1651                    DisplayRS232Error ();
1652                }
1653
1654
1655 // Camera
1656
1657            GetCtrlVal (panel_handle, MAIN_GRAB_2, &test2); //check if the grab button is pressed
1658
1659            if(test2)
1660            {
1661            SetCtrlAttribute (panel_handle, MAIN_TIMER, ATTR_ENABLED, FALSE);
1662            // Stop the acquisition
1663            IMAQdxStopAcquisition (session);
1664            // Unconfigure the acquisition
1665            IMAQdxUnconfigureAcquisition (session);
1666            }
1667
1668            if(session)  //if the camera session is still running
1669            {
1670            IMAQdxCloseCamera (session); //close the session and set the value to 0
1671            session = 0;
1672            }
1673
1674            // Delete any image in memory
1675            if(images)  //if there are images from the manual sequential grab
1676            {
1677                for(i=0; i<imageArraySize; i++)
1678                    imaqDispose (images[i]);    //clear array completely
1679
1680                free(images);
1681                images = NULL;
1682            }
1683
1684            if(image_g)  //if there are images from the grab function
1685                {
1686                imaqDispose(image_g); //Dispose the images
1687                image_g = NULL;      //Set array to Null
1688                }
1689
1690            if(image)    //if there is an image from the snap function
1691            {
1692               imaqDispose(image);    //dispose the image
1693               image = NULL;          // set value to null
1694            }
1695
1696            // Delete the Cam Attributes array in memory
1697            if (cameraAttributes)
1698            {
1699                free(cameraAttributes);
1700                cameraAttributes = NULL;
1701            }
1702
1703
1704
1705
1706
1707
1708            QuitUserInterface (0);
1709            break;
1710        case EVENT_RIGHT_CLICK :
1711            DisplayHelp (QuitHelp);
1712            break;
1713        }
1714    return 0;
1715 }
1716
1717 /*---------------------------------------------------------------------------*/
1718 /* Display error information to the user.                                    */
1719 /*---------------------------------------------------------------------------*/
1720 void DisplayRS232Error (void)
1721 {
1722     char ErrorMessage[200];
1723     switch (RS232Error)
1724         {
1725         default :
1726             if (RS232Error < 0)
1727                 {
1728                 Fmt (ErrorMessage, "%s<RS232 error number %i", RS232Error);
1729                 MessagePopup ("RS232 Message", ErrorMessage);
1730                 }
1731             break;
1732         case 0  :
1733             MessagePopup ("RS232 Message", "No errors.");
1734             break;
1735         case -2 :
1736             Fmt (ErrorMessage, "%s", "Invalid port number (must be in the "
1737                                      "range 1 to 8).");
1738             MessagePopup ("RS232 Message", ErrorMessage);
1739             break;
1740         case -3 :
1741             Fmt (ErrorMessage, "%s", "No port is open.\n"
1742                  "Check COM Port setting in Configure.");
1743             MessagePopup ("RS232 Message", ErrorMessage);
1744             break;
1745         case -99 :
```

```
1746            Fmt (ErrorMessage, "%s", "Timeout error.\n\n"
1747                 "Either increase timeout value,\n"
1748                 "       check COM Port setting, or\n"
1749                 "       check device.");
1750            MessagePopup ("RS232 Message", ErrorMessage);
1751            break;
1752        }
1753 }
1754
1755 /*---------------------------------------------------------------------------*/
1756 /* Display help information to the user.                                     */
1757 /*---------------------------------------------------------------------------*/
1758 void DisplayHelp (int HelpId)
1759 {
1760     switch (HelpId)
1761         {
1762         case 1 :
1763             MessagePopup ("Quit Help",
1764                           "The Quit button closes the current COM port,\n"
1765                           "checks and displays any error messages, \n"
1766                           "and then exits this program.");
1767             break;
1768         case 2 :
1769             MessagePopup ("Input Queue Help",
1770                           "Specifies the size of the input queue for the "
1771                           "selected port.\n"
1772                           "Default Value:  512\n"
1773                           "Valid Range:    28 to 65516");
1774             break;
1775         }
1776 }
1777
1778 /* C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D */
1779 //                                                                           //
1780 //           DDD  DDD  EEEE       SSS  EEEE RRRR  V     V EEEE RRRR   SSS       //
1781 //           D  D D  D E         S     E    R   R V     V E    R   R S          //
1782 //           D  D D  D EEE        SSS  EEE  RRRR   V   V  EEE  RRRR   SSS       //
1783 //           D  D D  D E             S E    R R     V V   E    R R       S      //
1784 //           DDD  DDD  EEEE      SSSS  EEEE R  RR    V    EEEE R  RR SSSS       //
1785 //                                                                           //
1786 /* C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D C8888D */
1787
1788 int CVICALLBACK Mass1ConfigCallback (int panel, int control, int event,
1789         void *callbackData, int eventData1, int eventData2)
1790 {
1791     static int check_connection;
1792     static int check_disconnection;
1793
1794     switch (event)
1795     {
1796         case EVENT_COMMIT:
1797
1798             GetCtrlVal(config_handle, CONFIG_M1, &m1check);     // bring current position (1=ON, 0=OFF)
1799             GetCtrlVal(config_handle, CONFIG_P1, &p1check);     // bring current position of pump (1=ON, 0=OFF)
1800
1801 if (p1check == 1)
1802 {
1803             if (m1check == 1)
1804             {
1805                 /*---------------------------------------------------------------------------*/
1806                 /*                            ClientDDERead                                  */
1807                 /*                  Requests data for the server to send                     */
1808                 /*---------------------------------------------------------------------------*/
1809                 check_connection =  ConnectToDDEServer (&hConv, "FlowDDE", "C(1)", DDECallback, 0);
1810                     // &hConv - unsigned *conversation handle
1811                     // "FlowDDE" - server name, e.g. "Word" - for MicroSoft Word
1812                     // "C(1)" - channel number - set by FlowDDE. topicName
1813                     // DDECallback - call back Function, required by DDE protocol. This function can receive only two messages: D
    DE_DISCONNECT and DDE_DATAREADY
1814
1815                     /* More details can be found here -  http://zone.ni.com/reference/en-XX/help/370051T-01/cvi/libref/cviconnect
    toddeserver/ */
1816                     /* key word for searching @ ni.com: ConnectToDDEServer   */
1817
1818                 if (check_connection>=0)
1819                 {
1820                     SetCtrlVal (config_handle, CONFIG_M1_LED, 1);
1821                     M1DDE_connection=1;
1822                                 // In case of error during connection procedure - check DDE_Library_ERR_codes.txt
1823                 }
1824                 else MessagePopup ("","Connection to FlowDDE failure");
1825             }
1826             else //Mass flow meter should be disconnected from server//
1827             {
1828                /*---------------------------------------------------------------------------*/
1829                /*                       DisconnectFromDDEServer                             */
1830                /*                  Requests data for the server to send                     */
1831                /*---------------------------------------------------------------------------*/
1832                 if (M1DDE_connection == 1) //checking for connection
1833                 {
1834                     check_disconnection = DisconnectFromDDEServer (hConv);   // disconnect from DDE server
1835
1836                     if (check_disconnection>=0)
1837                     {
1838                         SetCtrlVal (config_handle, CONFIG_M1_LED, 0);            // set LED in "OFF" position
1839                         M1DDE_connection =0;
1840                     }
```

```
                        else MessagePopup ("","Disconnection from FlowDDE failure");
                }
                        else MessagePopup ("","There is no connection to FlowDDE");
            }
}
else
{
MessagePopup ("Warning","Pump #1 connection is required");
SetCtrlVal(config_handle, CONFIG_M1, 0);
}
    break;
    }
    return 0;
}

int CVICALLBACK Mass2ConfigCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    static int check_connection;
    static int check_disconnection;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(config_handle, CONFIG_M2, &m2check);
            GetCtrlVal(config_handle, CONFIG_P2, &p2check);
if (p2check == 1)
{
            if (m2check == 1)
            {
                check_connection =  ConnectToDDEServer (&hConv2, "FlowDDE2", "C(1)", DDECallback, 0);
                if (check_connection>=0)
                {
                    SetCtrlVal (config_handle, CONFIG_M2_LED, 1);
                    M2DDE_connection=1;
                }
                else MessagePopup ("","Connection to FlowDDE2 failure");
            }
            else
            {
                if (M2DDE_connection == 1) //checking for connection
                {
                    check_disconnection = DisconnectFromDDEServer (hConv2);   // disconnect from DDE server

                    if (check_disconnection>=0)
                    {
                        SetCtrlVal (config_handle, CONFIG_M2_LED, 0);            // set LED in "OFF" position
                        M2DDE_connection =0;
                    }
                        else MessagePopup ("","Disconnection from FlowDDE2 failure");
                }
                        else MessagePopup ("","There is no connection to FlowDDE2");
            }
}
else
{
MessagePopup ("Warning","Pump #2 connection is required");
SetCtrlVal(config_handle, CONFIG_M2, 0);
}
    break;
    }
    return 0;
}

int CVICALLBACK Mass3ConfigCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    static int check_connection;
    static int check_disconnection;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(config_handle, CONFIG_M3, &m3check);

            if (m3check == 1)
            {
                check_connection =  ConnectToDDEServer (&hConv3, "FlowDDE3", "C(1)", DDECallback, 0);
                if (check_connection>=0)
                {
                    SetCtrlVal (config_handle, CONFIG_M3_LED, 1);
                    M3DDE_connection=1;
                }
                else MessagePopup ("","Connection to FlowDDE3 failure");

            }
            else
            {
                if (M3DDE_connection == 1) //checking for connection
                {
                    check_disconnection = DisconnectFromDDEServer (hConv3);   // disconnect from DDE server

                    if (check_disconnection>=0)
                    {
                        SetCtrlVal (config_handle, CONFIG_M3_LED, 0);            // set LED in "OFF" position
                        M3DDE_connection =0;
                    }
                        else MessagePopup ("","Disconnection from FlowDDE3 failure");
                }
```

```
                        else MessagePopup ("","There is no connection to FlowDDE3");
            }
    break;
    }
    return 0;
}

int CVICALLBACK Mass4ConfigCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    static int check_connection;
    static int check_disconnection;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(config_handle, CONFIG_M4, &m4check);

            if (m4check == 1)
            {
                check_connection =  ConnectToDDEServer (&hConv4, "FlowDDE4", "C(1)", DDECallback, 0);
                if (check_connection>=0)
                {
                    SetCtrlVal (config_handle, CONFIG_M4_LED, 1);
                    M4DDE_connection=1;
                }
                else MessagePopup ("","Connection to FlowDDE4 failure");
            }
            else
            {
                if (M4DDE_connection == 1) //checking for connection
                {
                    check_disconnection = DisconnectFromDDEServer (hConv4);   // disconnect from DDE server

                    if (check_disconnection>=0)
                    {
                        SetCtrlVal (config_handle, CONFIG_M4_LED, 0);            // set LED in "OFF" position
                        M4DDE_connection =0;
                    }
                        else MessagePopup ("","Disconnection from FlowDDE4 failure");
                }
                else MessagePopup ("","There is no connection to FlowDDE4");
            }
    break;
    }
    return 0;
}

int CVICALLBACK BPRConfigCallback (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    static int check_connection;
    static int check_disconnection;
    switch (event)
    {
        case EVENT_COMMIT:
            GetCtrlVal(config_handle, CONFIG_BPR, &bprcheck);

            if (bprcheck == 1)
            {
                check_connection =  ConnectToDDEServer (&hConv5, "FlowDDE5", "C(1)", DDECallback, 0);
                if (check_connection>=0)
                {
                    SetCtrlVal (config_handle, CONFIG_BPR_LED, 1);
                    BPRDDE_connection=1;
                }
                else MessagePopup ("","Connection to FlowDDE5 failure");
            }
            else
            {
                if (BPRDDE_connection == 1) //checking for connection
                {
                    check_disconnection = DisconnectFromDDEServer (hConv5);   // disconnect from DDE server

                    if (check_disconnection>=0)
                    {
                        SetCtrlVal (config_handle, CONFIG_BPR_LED, 0);           // set LED in "OFF" position
                        BPRDDE_connection =0;
                    }
                        else MessagePopup ("","Disconnection from FlowDDE5 failure");
                }
                else MessagePopup ("","There is no connection to FlowDDE5");
            }
    break;
    }
    return 0;
}

/*---------------------------------------------------------------------------*/
//This function processes messages as the client of the DDE Server (FlowDDE).
//If for some reason FlowDDE needs to shut don, it will issue wdisconnect messages to its clients
//and this function processes messages from FlowDDE to its clients.
/*---------------------------------------------------------------------------*/
int DDECallback (unsigned handle, char *topicName,
                        char *itemName, int xType, int dataFmt,
                        int dataSize, void *dataPtr,
                        void *callbackData)
{
```

```
2035     switch (xType) {
2036         case DDE_DISCONNECT:
2037             SetInputMode (config_handle, CONFIG_M1_LED, 0);
2038             MessagePopup("", "FlowDDE has shut down the connection");
2039             break;
2040     }
2041
2042     return 1;
2043 }
2044
2045 int CVICALLBACK P1ActivateCallBack (int panel, int control, int event,
2046         void *callbackData, int eventData1, int eventData2)
2047 {
2048
2049
2050 switch (event)
2051 {
2052 case EVENT_COMMIT:
2053 GetCtrlVal (panel_handle, MAIN_P1_ONOFF, &test);
2054         if (p1check == 1)
2055         {
2056             if (Pumps_quitflag == 1)
2057             {
2058              CmtGetLock     (Pumps_lockHandle);
2059                 ActivatePump1 ();
2060              CmtReleaseLock (Pumps_lockHandle);
2061             }
2062             else ActivatePump1 ();
2063         }
2064         else
2065         {
2066         MessagePopup ("Warning","Connection to Pump #1 is required");
2067         Delay(0.05);
2068         SetCtrlVal(panel_handle, MAIN_P1_ONOFF, 0);
2069         }
2070     break;
2071     }
2072     return 0;
2073 }
2074
2075 void ActivatePump1 (void)
2076 {
2077 int i;
2078 double probe_temp;
2079 char temp [NUM];
2080                     if(test==1)
2081                     {
2082
2083                             EnableP1PanelControls (0);
2084                             SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_1, ATTR_VISIBLE, 1);
2085                             SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_1, ATTR_VISIBLE, 1);
2086                             SetCtrlVal (panel_handle, MAIN_P1_ONOFF_LED, 1);
2087
2088                             ////////////
2089                             panel_name=panel_handle;
2090                             CP=P1_comport;
2091                             text_panel_name= MAIN_WAS_FR_P_1;
2092                             ////////////
2093
2094                             FlushOutQ (CP);
2095                             FlushInQ (CP);
2096
2097                             send_data[0] = '\0';
2098                             Fmt (send_data, "%s<F?\r");
2099                             stringsize = StringLength (send_data);
2100
2101                             bytes_sent = ComWrt (CP, send_data, stringsize);
2102                             Delay(0.05); /* 0.05  */
2103
2104                             read_cnt = GetInQLen (CP);
2105
2106                             read_data[0] = '\0';
2107                             read_term = 13;
2108
2109                             bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
2110                             CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
2111
2112                                 for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
2113                                     temp[i-2]=tbox_read_data[i];
2114                                     temp[i-2]='\0';         //close by '\0'
2115
2116                                 probe_temp = atof(temp);
2117                             ProcessSystemEvents ();
2118                                 SetCtrlVal (panel_name, text_panel_name, probe_temp);
2119
2120                             PFR11[0] = probe_temp;      // Log value in PFR11 Array
2121
2122                             RS232Error = ReturnRS232Err ();
2123                             if (RS232Error)
2124                                 DisplayRS232Error ();
2125
2126                             send_data[0] = '\0';
2127
2128                             SetCtrlVal(panel_handle, MAIN_LOG_P1, 1);   //activate logging pressure
2129                             GetCtrlVal(panel_handle, MAIN_LOG_P1, &Is_P1);
2130                     }
2131                     else
```

```
                    {
                        SetCtrlVal(panel_handle, MAIN_LOG_P1, 0);       //deactivate logging pressure
                        GetCtrlVal(panel_handle, MAIN_LOG_P1, &Is_P1);  //save new value


                        EnableP1PanelControls (1);  //dim all panels
                        SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_1, ATTR_VISIBLE, 0);
                        SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_1, ATTR_VISIBLE, 0);
                        SetCtrlVal (panel_handle, MAIN_P1_ONOFF_LED, 0);

                            ////////////
                            panel_name=panel_handle;
                            CP=P1_comport;
                            PTEXT_panel_name= MAIN_STATUS_P_1;
                            ////////////

                        send_data[0] = '\0';
                        Fmt (send_data, "%s<OFF\r");
                        SendReadPump ();
                        PFR11[0] = '\0';        // Log value in PFR11 Array
                    }
return;
}

//*******************************************************
// Load_Config      : Load variables from config file
//                  : Update screen from variables
// Orgignally made by DSN
//*******************************************************

 void CVICALLBACK Load_Config (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    LoadConfig();
    LoadVars();
    Update_Graphics();

}

//*******************************************************
// Save_Config      : Save screen to variables
//                  : Save variables to config file
// Orgignally made by DSN
//*******************************************************

void CVICALLBACK Save_Config (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    SaveVars();
    SaveConfig(0);
}

//*******************************************************
// Save_Config_As   : Save screen to variables
//                  : Save variables to config file
//                  : Allow the config file to be selected
// Orgignally made by DSN
//*******************************************************

void CVICALLBACK Save_Config_As (int menuBar, int menuItem, void *callbackData,
        int panel)
{
    SaveVars();
    SaveConfig(1);
}

void CVICALLBACK ExitCallback (int menuBar, int menuItem, void *callbackData, int panel) //dpne
{
    DisplayPanel (s_handle);
}

void CVICALLBACK Com_Setup (int menuBar, int menuItem, void *callbackData, int panel) //done
{
     DisplayPanel(config_handle);

      /*  If user already has done configuration, then
          display those new parameters.  If entering
          configuration for 1st time, set config_flag
          and use default settings.
       */
     if (config_flag)    /* Configuration done at least once.*/
     SetConfigParms ();
     else                /* 1st time.*/
         config_flag = 1;

}

void CVICALLBACK About_Show (int menuBar, int menuItem, void *callbackData,  //Display About panel
        int panel)
{
         DisplayPanel(a_handle);
}

void CVICALLBACK Graph_set (int menuBar, int menuItem, void *callbackData,   //Display Graph panel
        int panel)
{
         DisplayPanel(g handle);
```

```
}

//////////////////////////////////////
////////////SOLENOID VALVES //////////
////////////// OPEN - CLOSE ////////////
//////////////////////////////////////

int CVICALLBACK AV123CallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)      /// Build based on WriteDigChan.c
{

if( event==EVENT_COMMIT ) SWITCH_SOLENOID();

return 0;

}

/*
int         error=0;
TaskHandle  taskHandleAV=0;
char        chanAV[256];
uInt8       dataAV[8];
char        errBuffAV[2048]={'\0'};

    if( event==EVENT_COMMIT )
    {
        GetCtrlVal(config_handle,CONFIG_SOLCHANNEL,chanAV);
        GetCtrlVal(panel_handle,MAIN_AV1_ONOFF,&dataAV[0]);   //#1 solenoid valve  [inlet]
        GetCtrlVal(panel_handle,MAIN_AV3_ONOFF,&dataAV[1]);   //#2 solenoid valve  [outlet]
        GetCtrlVal(panel_handle,MAIN_AV2_ONOFF,&dataAV[2]);   //#3 solenoid valve  [dP valve]

        if  (dataAV[0] == 1) SetCtrlVal (panel_handle, MAIN_AV1_LED, 1);  //Show up status of AV-1
        else SetCtrlVal (panel_handle, MAIN_AV1_LED, 0);

        if  (dataAV[1] == 1) SetCtrlVal (panel_handle, MAIN_AV3_LED, 1);  //Show up status of AV-2
        else SetCtrlVal (panel_handle, MAIN_AV3_LED, 0);

        if  (dataAV[2] == 1) SetCtrlVal (panel_handle, MAIN_AV2_LED, 1);  //Show up status of AV-3
        else SetCtrlVal (panel_handle, MAIN_AV2_LED, 0);


        //********************************************
        // DAQmx Configure Code
        //********************************************
        SetWaitCursor(1);
        DAQmxErrChk (DAQmxCreateTask("",&taskHandleAV));
        DAQmxErrChk (DAQmxCreateDOChan(taskHandleAV,chanAV,"",DAQmx_Val_ChanForAllLines));

        //********************************************
        // DAQmx Start Code
        //********************************************
        DAQmxErrChk (DAQmxStartTask(taskHandleAV));

        //********************************************
        // DAQmx Write Code
        //********************************************
        DAQmxErrChk (DAQmxWriteDigitalLines(taskHandleAV,1,1,10.0,DAQmx_Val_GroupByChannel,dataAV,NULL,NULL));
    }

Error:
    SetWaitCursor(0);
    if( DAQmxFailed(error) )
        DAQmxGetExtendedErrorInfo(errBuffAV,2048);
    if( taskHandleAV!=0 ) {
        //********************************************
        // DAQmx Stop Code
        //********************************************
        DAQmxStopTask(taskHandleAV);
        DAQmxClearTask(taskHandleAV);
    }
    if( DAQmxFailed(error) )
        MessagePopup("DAQmx Error",errBuffAV);
    return 0;
*/

void SWITCH_SOLENOID (void)    //Function to switch solenoid valves
{
int         error=0;
TaskHandle  taskHandleAV=0;
char        chanAV[256];
uInt8       dataAV[8];
char        errBuffAV[2048]={'\0'};


        GetCtrlVal(config_handle,CONFIG_SOLCHANNEL,chanAV);
        GetCtrlVal(panel_handle,MAIN_AV1_ONOFF,&dataAV[0]);   //#1 solenoid valve  [inlet]

        dataAV[1]=dataAV[0]; //#2 solenoid valve  [outlet] if one set to close[open] - same for second
        SetCtrlVal(panel_handle, MAIN_AV3_ONOFF, dataAV[1]);  //#2 solenoid valve  [outlet]

                    //  GetCtrlVal(panel_handle,MAIN_AV3_ONOFF,&dataAV[1]);   //#2 solenoid valve  [outlet]

        GetCtrlVal(panel_handle,MAIN_AV2_ONOFF,&dataAV[2]);   //#3 solenoid valve  [dP valve]

        if  (dataAV[0] == 1) SetCtrlVal (panel_handle, MAIN_AV1_LED, 1);  //Show up status of AV-1
        else SetCtrlVal (panel_handle, MAIN_AV1_LED, 0);

```

```
2326         if  (dataAV[1] == 1) SetCtrlVal (panel_handle, MAIN_AV3_LED, 1);  //Show up status of AV-2
2327         else SetCtrlVal (panel_handle, MAIN_AV3_LED, 0);
2328
2329         if  (dataAV[2] == 1) SetCtrlVal (panel_handle, MAIN_AV2_LED, 1);  //Show up status of AV-3
2330         else SetCtrlVal (panel_handle, MAIN_AV2_LED, 0);
2331
2332
2333         /*********************************************/
2334         // DAQmx Configure Code
2335         /*********************************************/
2336         SetWaitCursor(1);
2337         DAQmxErrChk (DAQmxCreateTask("",&taskHandleAV));
2338         DAQmxErrChk (DAQmxCreateDOChan(taskHandleAV,chanAV,"",DAQmx_Val_ChanForAllLines));
2339
2340         /*********************************************/
2341         // DAQmx Start Code
2342         /*********************************************/
2343         DAQmxErrChk (DAQmxStartTask(taskHandleAV));
2344
2345         /*********************************************/
2346         // DAQmx Write Code
2347         /*********************************************/
2348         DAQmxErrChk (DAQmxWriteDigitalLines(taskHandleAV,1,1,10.0,DAQmx_Val_GroupByChannel,dataAV,NULL,NULL));
2349
2350
2351 Error:
2352     SetWaitCursor(0);
2353     if( DAQmxFailed(error) )
2354         DAQmxGetExtendedErrorInfo(errBuffAV,2048);
2355     if( taskHandleAV!=0 ) {
2356         /*********************************************/
2357         // DAQmx Stop Code
2358         /*********************************************/
2359         DAQmxStopTask(taskHandleAV);
2360         DAQmxClearTask(taskHandleAV);
2361     }
2362     if( DAQmxFailed(error) )
2363         MessagePopup("DAQmx Error",errBuffAV);
2364
2365 return;
2366 }
2367
2368 void MainPanelQuit(void)  //Function to close user interface
2369 {
2370 //  int i;
2371
2372     QuitUserInterface(0);
2373
2374 return;
2375 }
2376
2377 int CVICALLBACK SaveYES_MCP_Config (int panel, int control, int event,    //Shutdown program with sawing config
2378         void *callbackData, int eventData1, int eventData2)
2379 {
2380     switch (event)
2381     {
2382         case EVENT_COMMIT:
2383             SaveConfig(0);
2384             HidePanel (s_handle);
2385             MainPanelQuit();
2386             break;
2387         case EVENT_RIGHT_CLICK:
2388             break;
2389     }
2390     return 0;
2391 }
2392
2393 int CVICALLBACK SaveNO_MCP_Config (int panel, int control, int event,     //Shutdown program without saving config
2394         void *callbackData, int eventData1, int eventData2)
2395 {
2396     switch (event)
2397     {
2398         case EVENT_COMMIT:
2399             HidePanel (s_handle);
2400             MainPanelQuit();
2401             break;
2402         case EVENT_RIGHT_CLICK:
2403             break;
2404     }
2405     return 0;
2406 }
2407
2408 int CVICALLBACK SaveCANCEL_MCP_Config (int panel, int control, int event, //Return back to program without saving config
2409         void *callbackData, int eventData1, int eventData2)
2410 {
2411     switch (event)
2412     {
2413         case EVENT_COMMIT:
2414             HidePanel (s_handle);
2415             break;
2416         case EVENT_RIGHT_CLICK:
2417             break;
2418     }
2419     return 0;
2420 }
2421
2422 int CVICALLBACK AboutCallBack (int panel, int control, int event, //Close About panel
```

```
2423         void *callbackData, int eventData1, int eventData2)
2424 {
2425     switch (event)
2426     {
2427         case EVENT_COMMIT:
2428             HidePanel (a_handle);
2429             break;
2430         case EVENT_RIGHT_CLICK:
2431             break;
2432     }
2433     return 0;
2434 }
2435
2436 int CVICALLBACK CloseGCallBack (int panel, int control, int event, //Close g_handle panel
2437         void *callbackData, int eventData1, int eventData2)
2438 {
2439     switch (event)
2440     {
2441         case EVENT_COMMIT:
2442             HidePanel (g_handle);
2443             break;
2444         case EVENT_RIGHT_CLICK:
2445             break;
2446     }
2447     return 0;
2448 }
2449
2450 int CVICALLBACK SETUP_G (int panel, int control, int event,  //Setup Graph Scaling to Match the Graph Panel Values
2451         void *callbackData, int eventData1, int eventData2)
2452 {
2453     switch (event)
2454     {
2455         case EVENT_COMMIT:
2456             GraphSetup(); //run settings for Graph
2457             break;
2458         case EVENT_RIGHT_CLICK:
2459             break;
2460     }
2461     return 0;
2462 }
2463
2464 //****************************************************************************
2465 // GraphSetup       : Setup Graph Scaling to Match the Graph Panel Values
2466 // build based on DSN_GraphSetup with additional function SetAxisScalingMode
2467 //****************************************************************************
2468
2469 void GraphSetup(void)
2470 {
2471
2472 // Update the variables from the panels
2473 SaveVars();
2474
2475                 //Read Graph #1
2476                     if(Y_Mode_G_1)  // Check for Auto mode
2477                        {
2478                        SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_1,  ATTR_LABEL_TEXT, "Auto");
2479                        SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_1,   ATTR_DIMMED, 1);
2480                        SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_1,   ATTR_DIMMED, 1);
2481
2482                        SetAxisScalingMode (panel_handle, MAIN_G_1, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_1);
2483                        SetAxisScalingMode (panel_handle, MAIN_G_1, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2484                        }
2485                     else
2486                         {
2487                         SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_1, ATTR_LABEL_TEXT, "Fixed");
2488                         SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_1,  ATTR_DIMMED, 0);
2489                         SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_1,  ATTR_DIMMED, 0);
2490
2491                         SetAxisScalingMode (panel_handle, MAIN_G_1, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_1);
2492                         SetAxisScalingMode (panel_handle, MAIN_G_1, VAL_LEFT_YAXIS, VAL_MANUAL, Y_Min_G_1, Y_Max_G_1);
2493                         }
2494
2495                         GraphName(plotVar_G_1, MAIN_G_1,G_SETUP_G1_L_1,G_SETUP_G1_T_1,
2496                                                         G_SETUP_G1_L_2,G_SETUP_G1_T_2,
2497                                                         G_SETUP_G1_L_3,G_SETUP_G1_T_3,
2498                                                         G_SETUP_G1_L_4,G_SETUP_G1_T_4);
2499                 //Read Graph #2
2500                     if(Y_Mode_G_2)  // Check for Auto mode
2501                        {
2502                        SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_2,  ATTR_LABEL_TEXT, "Auto");
2503                        SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_2,   ATTR_DIMMED, 1);
2504                        SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_2,   ATTR_DIMMED, 1);
2505
2506                        SetAxisScalingMode (panel_handle, MAIN_G_2, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_2);
2507                        SetAxisScalingMode (panel_handle, MAIN_G_2, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2508                        }
2509                     else
2510                         {
2511                         SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_2, ATTR_LABEL_TEXT, "Fixed");
2512                         SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_2,  ATTR_DIMMED, 0);
2513                         SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_2,  ATTR_DIMMED, 0);
2514
2515                         SetAxisScalingMode (panel_handle, MAIN_G_2, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_2);
2516                         SetAxisScalingMode (panel_handle, MAIN_G_2, VAL_LEFT_YAXIS, VAL_MANUAL, Y_Min_G_2, Y_Max_G_2);
2517
2518                         }
2519
```

```
2520                    GraphName(plotVar_G_2, MAIN_G_2,G_SETUP_G2_L_1,G_SETUP_G2_T_1,
2521                                                    G_SETUP_G2_L_2,G_SETUP_G2_T_2,
2522                                                    G_SETUP_G2_L_3,G_SETUP_G2_T_3,
2523                                                    G_SETUP_G2_L_4,G_SETUP_G2_T_4);
2524                //Read Graph #3
2525                    if(Y_Mode_G_3)  // Check for Auto mode
2526                       {
2527                       SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_3,  ATTR_LABEL_TEXT, "Auto");
2528                       SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_3,   ATTR_DIMMED, 1);
2529                       SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_3,   ATTR_DIMMED, 1);
2530
2531                       SetAxisScalingMode (panel_handle, MAIN_G_3, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_3);
2532                       SetAxisScalingMode (panel_handle, MAIN_G_3, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2533                       }
2534                    else
2535                        {
2536                        SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_3, ATTR_LABEL_TEXT, "Fixed");
2537                        SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_3,  ATTR_DIMMED, 0);
2538                        SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_3,  ATTR_DIMMED, 0);
2539
2540                        SetAxisScalingMode (panel_handle, MAIN_G_3, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_3);
2541                        SetAxisScalingMode (panel_handle, MAIN_G_3, VAL_LEFT_YAXIS, VAL_MANUAL, Y_Min_G_3, Y_Max_G_3);
2542                        }
2543
2544                        GraphName(plotVar_G_3, MAIN_G_3,G_SETUP_G3_L_1,G_SETUP_G3_T_1,
2545                                                G_SETUP_G3_L_2,G_SETUP_G3_T_2,
2546                                                G_SETUP_G3_L_3,G_SETUP_G3_T_3,
2547                                                G_SETUP_G3_L_4,G_SETUP_G3_T_4);
2548                //Read Graph #4
2549                    if(Y_Mode_G_4)  // Check for Auto mode
2550                       {
2551                       SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_4,  ATTR_LABEL_TEXT, "Auto");
2552                       SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_4,   ATTR_DIMMED, 1);
2553                       SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_4,   ATTR_DIMMED, 1);
2554
2555                       SetAxisScalingMode (panel_handle, MAIN_G_4, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_4);
2556                       SetAxisScalingMode (panel_handle, MAIN_G_4, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2557                       }
2558                    else
2559                        {
2560                        SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_4, ATTR_LABEL_TEXT, "Fixed");
2561                        SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_4,  ATTR_DIMMED, 0);
2562                        SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_4,  ATTR_DIMMED, 0);
2563
2564                        SetAxisScalingMode (panel_handle, MAIN_G_4, VAL_BOTTOM_XAXIS, VAL_MANUAL, 0.0, X_Range_G_4);
2565                        SetAxisScalingMode (panel_handle, MAIN_G_4, VAL_LEFT_YAXIS, VAL_MANUAL, Y_Min_G_4, Y_Max_G_4);

2566                        }
2567
2568                        GraphName(plotVar_G_4, MAIN_G_4,G_SETUP_G4_L_1,G_SETUP_G4_T_1,
2569                                                G_SETUP_G4_L_2,G_SETUP_G4_T_2,
2570                                                G_SETUP_G4_L_3,G_SETUP_G4_T_3,
2571                                                G_SETUP_G4_L_4,G_SETUP_G4_T_4);
2572
2573                //Read Graph #5
2574                    if(X_Mode_G_5)  // Check for Auto mode X axis
2575                       {
2576                       SetCtrlAttribute (g_handle, G_SETUP_X_Mode_G_5,  ATTR_LABEL_TEXT, "Auto");
2577                       SetCtrlAttribute (g_handle, G_SETUP_X_Min_G_5,   ATTR_DIMMED, 1);
2578                       SetCtrlAttribute (g_handle, G_SETUP_X_Max_G_5,   ATTR_DIMMED, 1);
2579                       SetAxisScalingMode (extra_g_handle, EXTRA_G_G_5, VAL_BOTTOM_XAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2580                       }
2581                    else
2582                        {
2583                        SetCtrlAttribute (g_handle, G_SETUP_X_Mode_G_5, ATTR_LABEL_TEXT, "Fixed");
2584                        SetCtrlAttribute (g_handle, G_SETUP_X_Min_G_5,  ATTR_DIMMED, 0);
2585                        SetCtrlAttribute (g_handle, G_SETUP_X_Max_G_5,  ATTR_DIMMED, 0);
2586                        SetAxisScalingMode (extra_g_handle, EXTRA_G_G_5, VAL_BOTTOM_XAXIS, VAL_MANUAL, X_Min_G_5, X_Max_G_5);

2587                        }
2588
2589                    if(Y_Mode_G_5)  // Check for Auto mode
2590                       {
2591                       SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_5,  ATTR_LABEL_TEXT, "Auto");
2592                       SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_5,   ATTR_DIMMED, 1);
2593                       SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_5,   ATTR_DIMMED, 1);
2594                       SetAxisScalingMode (extra_g_handle, EXTRA_G_G_5, VAL_LEFT_YAXIS, VAL_AUTOSCALE, 0.0, 0.0);
2595                       }
2596                    else
2597                        {
2598                        SetCtrlAttribute (g_handle, G_SETUP_Y_Mode_G_5, ATTR_LABEL_TEXT, "Fixed");
2599                        SetCtrlAttribute (g_handle, G_SETUP_Y_Min_G_5,  ATTR_DIMMED, 0);
2600                        SetCtrlAttribute (g_handle, G_SETUP_Y_Max_G_5,  ATTR_DIMMED, 0);
2601                        SetAxisScalingMode (extra_g_handle, EXTRA_G_G_5, VAL_LEFT_YAXIS, VAL_MANUAL, Y_Min_G_5, Y_Max_G_5);

2602                        }
2603
2604                        GraphName(plotVar_G_5, EXTRA_G_G_5,  G_SETUP_G5_L_1,G_SETUP_G5_T_1,
2605                                                G_SETUP_G5_L_2,G_SETUP_G5_T_2,
2606                                                G_SETUP_G5_L_3,G_SETUP_G5_T_3,
2607                                                G_SETUP_G5_L_4,G_SETUP_G5_T_4);
2608                return;
2609                }
2610
2611                //******************************************************
2612                // GraphName        : Setup Graph Appearance
2613                //******************************************************
```

```
void GraphName(int Val, int GRAPH, int L1, int T1, int L2, int T2,
                                   int L3, int T3, int L4, int T4)
{
    switch(Val)
        {
        case -2:
             SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_YNAME, "Not Plotting");
             SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_RIGHT_YAXIS);
             SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_XNAME, "");

             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T1, "");
             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T2, "");
             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
             SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T4, "");
             break;
        case -1:
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Not Plotting");
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_RIGHT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "");
             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T1, "");
             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T2, "");
             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
             SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T4, "");
             break;
        case 0: // Pressure [Pump#1 & Inlet TP-2]
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_ACTIVE_XAXIS,VAL_BOTTOM_XAXIS);
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_XNAME, "Time [s]");

             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_handle, T1, "Pressure Pump #1");
             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
                   SetCtrlVal (g_handle, T2, "Pressure Inlet TP-2");

             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
             SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T4, "");
             break;
        case 1: // Pressure [Pump#2 & Inlet TP-2]
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_ACTIVE_XAXIS,VAL_BOTTOM_XAXIS);
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_XNAME, "Time [s]");

             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_handle, T1, "Pressure Pump #2");
             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
                   SetCtrlVal (g_handle, T2, "Pressure Inlet TP-2");

             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
             SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T4, "");
             break;
        case 2: // Pressure [Pump#3 & Inlet TP-2]
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_ACTIVE_XAXIS,VAL_BOTTOM_XAXIS);
             SetCtrlAttribute(panel_handle, GRAPH, ATTR_XNAME, "Time [s]");

             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_handle, T1, "Pressure Pump #3");
             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
                   SetCtrlVal (g_handle, T2, "Pressure Inlet TP-2");

             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
             SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T4, "");
             break;

        case 3: // Mass flow rate [M#1] WATER
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Mass flow rate [kg/min]");
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
             SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
        //   SetCtrlAttribute (panel_handle, GRAPH, ATTR_XDIVISIONS, 3); number of divisions

             SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
                   SetCtrlVal (g_handle, T1, "Mass flow rate [M#1] WATER");

             SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T2, "");
             SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
                   SetCtrlVal (g_handle, T3, "");
```

```
2711                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2712                              SetCtrlVal (g_handle, T4, "");
2713                        break;
2714
2715                  case 4: // Mass flow rate [M#2] OIL
2716                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2717                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Mass flow rate [kg/min]");
2718                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2719                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2720
2721                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2722                              SetCtrlVal (g_handle, T1, "Mass flow rate [M#2] OIL");
2723
2724                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2725                              SetCtrlVal (g_handle, T2, "");
2726                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2727                              SetCtrlVal (g_handle, T3, "");
2728                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2729                              SetCtrlVal (g_handle, T4, "");
2730                        break;
2731
2732                  case 5: // Mass flow rate [M#3] EMULSION
2733                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2734                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Mass flow rate [kg/min]");
2735                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2736                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2737
2738                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2739                              SetCtrlVal (g_handle, T1, "Mass flow rate [M#3] EMULSION");
2740
2741                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2742                              SetCtrlVal (g_handle, T2, "");
2743                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2744                              SetCtrlVal (g_handle, T3, "");
2745                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2746                              SetCtrlVal (g_handle, T4, "");
2747                        break;
2748
2749                  case 6: // Mass flow rate [M#4] Mixture
2750                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2751                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Mass flow rate [kg/min]");
2752                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2753                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2754
2755                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2756                              SetCtrlVal (g_handle, T1, "Mass flow rate [M#4] Mixture [outlet]");
2757
2758                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2759                              SetCtrlVal (g_handle, T2, "");
2760                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2761                              SetCtrlVal (g_handle, T3, "");
2762                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2763                              SetCtrlVal (g_handle, T4, "");
2764                        break;
2765
2766                  case 7: // Density [M#1] WATER
2767                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2768                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Density [gm/cm3]");
2769                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2770                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2771
2772                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2773                              SetCtrlVal (g_handle, T1, "Density [M#1] WATER");
2774
2775                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2776                              SetCtrlVal (g_handle, T2, "");
2777                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2778                              SetCtrlVal (g_handle, T3, "");
2779                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2780                              SetCtrlVal (g_handle, T4, "");
2781                        break;
2782
2783                  case 8: // Density [M#2] OIL
2784                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2785                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Density [gm/cm3]");
2786                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2787                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2788
2789                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2790                              SetCtrlVal (g_handle, T1, "Density [M#2] OIL");
2791
2792                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2793                              SetCtrlVal (g_handle, T2, "");
2794                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2795                              SetCtrlVal (g_handle, T3, "");
2796                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2797                              SetCtrlVal (g_handle, T4, "");
2798                        break;
2799
2800                  case 9: // Density [M#3] EMULSION
2801                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2802                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Density [gm/cm3]");
2803                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2804                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2805
2806                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2807                              SetCtrlVal (g_handle, T1, "Density [M#3] EMULSION");
```

```
2808
2809                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2810                              SetCtrlVal (g_handle, T2, "");
2811                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2812                              SetCtrlVal (g_handle, T3, "");
2813                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2814                              SetCtrlVal (g_handle, T4, "");
2815                        break;
2816
2817                  case 10: // Density [M#4] Mixture
2818                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2819                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Density [gm/cm3]");
2820                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2821                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2822
2823                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2824                              SetCtrlVal (g_handle, T1, "Density [M#4] Mixture [outlet]");
2825
2826                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2827                              SetCtrlVal (g_handle, T2, "");
2828                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2829                              SetCtrlVal (g_handle, T3, "");
2830                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2831                              SetCtrlVal (g_handle, T4, "");
2832                        break;
2833
2834                  case 11: // Pressure TP-2
2835                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2836                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
2837                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2838                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2839
2840                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2841                              SetCtrlVal (g_handle, T1, "Pressure Inlet TP-2");
2842
2843                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2844                              SetCtrlVal (g_handle, T2, "");
2845                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2846                              SetCtrlVal (g_handle, T3, "");
2847                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2848                              SetCtrlVal (g_handle, T4, "");
2849                        break;
2850
2851                  case 12: // Pressure dP1
2852                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2853                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
2854                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2855                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2856
2857                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2858                              SetCtrlVal (g_handle, T1, "Differential pressure dP1");
2859
2860                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2861                              SetCtrlVal (g_handle, T2, "");
2862                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2863                              SetCtrlVal (g_handle, T3, "");
2864                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2865                              SetCtrlVal (g_handle, T4, "");
2866                        break;
2867
2868                  case 13: // Pressure Delta
2869                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2870                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
2871                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2872                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2873
2874                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2875                              SetCtrlVal (g_handle, T1, "Differential pressure TP-2 - BPR1");
2876
2877                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2878                              SetCtrlVal (g_handle, T2, "");
2879                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2880                              SetCtrlVal (g_handle, T3, "");
2881                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2882                              SetCtrlVal (g_handle, T4, "");
2883                        break;
2884
2885                  case 14: // Pressure Delta vs dP1
2886                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2887                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
2888                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2889                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2890
2891                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2892                              SetCtrlVal (g_handle, T1, "Differential pressure TP-2 - BPR1");
2893                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
2894                              SetCtrlVal (g_handle, T2, "Differential pressure dP1");
2895
2896                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2897                              SetCtrlVal (g_handle, T3, "");
2898                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2899                              SetCtrlVal (g_handle, T4, "");
2900                        break;
2901
2902                  case 15: // Water inj
2903                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2904                        SetCtrlAttribute (panel_handle, GRAPH, ATTR YNAME, "Volume [cc]");
```

```
2905                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2906                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2907
2908                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2909                              SetCtrlVal (g_handle, T1, "Water injected");
2910
2911                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2912                              SetCtrlVal (g_handle, T2, "");
2913                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2914                              SetCtrlVal (g_handle, T3, "");
2915                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2916                              SetCtrlVal (g_handle, T4, "");
2917                        break;
2918
2919                  case 16: // Oil inj
2920                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2921                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Volume [cc]");
2922                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2923                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2924
2925                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2926                              SetCtrlVal (g_handle, T1, "Oil injected");
2927
2928                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2929                              SetCtrlVal (g_handle, T2, "");
2930                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2931                              SetCtrlVal (g_handle, T3, "");
2932                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2933                              SetCtrlVal (g_handle, T4, "");
2934                        break;
2935
2936                  case 17: // Emulsion inj
2937                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2938                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Volume [cc]");
2939                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2940                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2941
2942                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2943                              SetCtrlVal (g_handle, T1, "Emulsion injected");
2944
2945                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2946                              SetCtrlVal (g_handle, T2, "");
2947                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2948                              SetCtrlVal (g_handle, T3, "");
2949                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2950                              SetCtrlVal (g_handle, T4, "");
2951                        break;
2952
2953                  case 18: // Temperature water
2954                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2955                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Temperature [degC]");
2956                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2957                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2958
2959                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2960                              SetCtrlVal (g_handle, T1, "Water");
2961                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_YELLOW);
2962                              SetCtrlVal (g_handle, T2, "Oil");
2963                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_DK_YELLOW);
2964                              SetCtrlVal (g_handle, T3, "Emulsion");
2965                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_DK_MAGENTA);
2966                              SetCtrlVal (g_handle, T4, "Mixture");
2967                        break;
2968
2969                  case 19: // Water Flooding
2970                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2971                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Volume [cc]");
2972                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2973                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2974
2975                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2976                              SetCtrlVal (g_handle, T1, "Water flooding");
2977
2978                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2979                              SetCtrlVal (g_handle, T2, "");
2980                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2981                              SetCtrlVal (g_handle, T3, "");
2982                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2983                              SetCtrlVal (g_handle, T4, "");
2984                        break;
2985
2986                  case 20: // Water Flooding PV
2987                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
2988                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Volume [PV]");
2989                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
2990                        SetCtrlAttribute (panel_handle, GRAPH, ATTR_XNAME, "Time [s]");
2991
2992                        SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
2993                              SetCtrlVal (g_handle, T1, "Water flooding");
2994
2995                        SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2996                              SetCtrlVal (g_handle, T2, "");
2997                        SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
2998                              SetCtrlVal (g_handle, T3, "");
2999                        SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3000                              SetCtrlVal (g_handle, T4, "");
3001                        break;
```

```
3002
3003                    case 21: // WaterIN2PV vs DP1
3004                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
3005                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
3006                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
3007                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_XNAME, "Volume inj [PV]");
3008
3009                         SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
3010                               SetCtrlVal (g_handle, T1, "Differential pressure DP-1");
3011
3012                         SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3013                               SetCtrlVal (g_handle, T2, "");
3014                         SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3015                               SetCtrlVal (g_handle, T3, "");
3016                         SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3017                               SetCtrlVal (g_handle, T4, "");
3018                         break;
3019
3020                    case 22: // WaterIN2PV vs DP1
3021                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
3022                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
3023                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
3024                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_XNAME, "Volume inj [PV]");
3025
3026                         SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
3027                               SetCtrlVal (g_handle, T1, "REF Differential pressure Delta");
3028
3029                         SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3030                               SetCtrlVal (g_handle, T2, "");
3031                         SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3032                               SetCtrlVal (g_handle, T3, "");
3033                         SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3034                               SetCtrlVal (g_handle, T4, "");
3035                         break;
3036
3037                    case 23: // WaterIN2PV vs DP1
3038                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
3039                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_YNAME, "Pressure [psi]");
3040                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_ACTIVE_XAXIS, VAL_BOTTOM_XAXIS);
3041                         SetCtrlAttribute (extra_g_handle, GRAPH, ATTR_XNAME, "Volume inj [PV]");
3042
3043                         SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_RED);
3044                               SetCtrlVal (g_handle, T1, "REF Differential pressure Delta");
3045                         SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_WHITE);
3046                               SetCtrlVal (g_handle, T2, "Differential pressure DP-1");
3047
3048                         SetCtrlAttribute (g_handle, L3, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3049                               SetCtrlVal (g_handle, T3, "");
3050                         SetCtrlAttribute (g_handle, L4, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3051                               SetCtrlVal (g_handle, T4, "");
3052                         break;
3053
3054                    case 24: // Experiment Stage
3055                         SetCtrlAttribute (panel_handle, GRAPH, ATTR_ACTIVE_YAXIS, VAL_LEFT_YAXIS);
3056
3057                         SetCtrlAttribute (panel_handle, GRAPH, ATTR_YNAME, "Experiment Stage");
3058                         SetCtrlAttribute (g_handle, L1, ATTR_FRAME_COLOR, VAL_LT_GRAY);
3059                               SetCtrlVal (g_handle, T1, "0 (Temp), 1 (Active), 2 (Thixo.)");
3060                               SetCtrlAttribute (panel_handle, GRAPH, ATTR_YDIVISIONS, 3);
3061                         SetCtrlAttribute (g_handle, L2, ATTR_FRAME_COLOR, VAL_RED);
3062                               SetCtrlVal (g_handle, T2, "");
3063                    }
3064               return;
3065               }
3066
3067               //*********************************************************
3068               // Save_Vars    : Save screen to variables
3069               // build based on DSN_Save_Vars
3070               //*********************************************************
3071               void SaveVars(void)
3072               { // int i;
3073
3074
3075               // MAIN PANEL
3076
3077
3078
3079                   // COM Ports
3080                   GetCtrlVal(panel_handle, MAIN_LOG_P1,           &Is_P1);
3081                   GetCtrlVal(panel_handle, MAIN_LOG_P2,           &Is_P2);
3082                   GetCtrlVal(panel_handle, MAIN_LOG_P3,           &Is_P3);
3083                   // Mass Flow Meters
3084                   GetCtrlVal(panel_handle, MAIN_LOG_M1,           &Is_M1);
3085                   GetCtrlVal(panel_handle, MAIN_LOG_M2,           &Is_M2);
3086                   GetCtrlVal(panel_handle, MAIN_LOG_M3,           &Is_M3);
3087                   GetCtrlVal(panel_handle, MAIN_LOG_M4,           &Is_M4);
3088
3089
3090                   // Back Pressure Regulator
3091                   GetCtrlVal(panel_handle, MAIN_LOG_BPR,      &Is_BPR);
3092                   // Main Parameters
3093                   GetCtrlVal(panel_handle, MAIN_LOG_Graph,        &Is_Graph);
3094
3095                   GetCtrlVal(panel_handle, MAIN_LOG_DAQ,      &Is_DAQ);
3096
3097                   GetCtrlVal(panel_handle, MAIN_LOG_LOGtoFile,    &Is_Log);
3098                   // Equipment Setup
```

```
3099
3100
3101                // GRAPH PANEL
3102                    //Read Graph #1
3103                    GetCtrlVal( g_handle,   G_SETUP_Var_G_1,        &plotVar_G_1);
3104                    GetCtrlVal( g_handle,   G_SETUP_X_Range_G_1,    &X_Range_G_1);
3105                    GetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_1,     &Y_Mode_G_1);
3106                    GetCtrlVal( g_handle,   G_SETUP_Y_Min_G_1,      &Y_Min_G_1);
3107                    GetCtrlVal( g_handle,   G_SETUP_Y_Max_G_1,      &Y_Max_G_1);
3108                    if (Y_Min_G_1 >= Y_Max_G_1)
3109                          Y_Min_G_1 = Y_Max_G_1-0.5;
3110                          SetCtrlVal( g_handle,     G_SETUP_Y_Min_G_1,      Y_Min_G_1);
3111                    //Read Graph #2
3112                    GetCtrlVal( g_handle,   G_SETUP_Var_G_2,        &plotVar_G_2);
3113                    GetCtrlVal( g_handle,   G_SETUP_X_Range_G_2,    &X_Range_G_2);
3114                    GetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_2,     &Y_Mode_G_2);
3115                    GetCtrlVal( g_handle,   G_SETUP_Y_Min_G_2,      &Y_Min_G_2);
3116                    GetCtrlVal( g_handle,   G_SETUP_Y_Max_G_2,      &Y_Max_G_2);
3117                    if (Y_Min_G_2 >= Y_Max_G_2)
3118                          Y_Min_G_2 = Y_Max_G_2-0.5;
3119                          SetCtrlVal( g_handle,     G_SETUP_Y_Min_G_2,      Y_Min_G_2);
3120                    //Read Graph #3
3121                    GetCtrlVal( g_handle,   G_SETUP_Var_G_3,        &plotVar_G_3);
3122                    GetCtrlVal( g_handle,   G_SETUP_X_Range_G_3,    &X_Range_G_3);
3123                    GetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_3,     &Y_Mode_G_3);
3124                    GetCtrlVal( g_handle,   G_SETUP_Y_Min_G_3,      &Y_Min_G_3);
3125                    GetCtrlVal( g_handle,   G_SETUP_Y_Max_G_3,      &Y_Max_G_3);
3126                    if (Y_Min_G_3 >= Y_Max_G_3)
3127                          Y_Min_G_3 = Y_Max_G_3-0.5;
3128                          SetCtrlVal( g_handle,     G_SETUP_Y_Min_G_3,      Y_Min_G_3);
3129                    //Read Graph #4
3130                    GetCtrlVal( g_handle,   G_SETUP_Var_G_4,        &plotVar_G_4);
3131                    GetCtrlVal( g_handle,   G_SETUP_X_Range_G_4,    &X_Range_G_4);
3132                    GetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_4,     &Y_Mode_G_4);
3133                    GetCtrlVal( g_handle,   G_SETUP_Y_Min_G_4,      &Y_Min_G_4);
3134                    GetCtrlVal( g_handle,   G_SETUP_Y_Max_G_4,      &Y_Max_G_4);
3135                    if (Y_Min_G_4 >= Y_Max_G_4)
3136                          Y_Min_G_4 = Y_Max_G_4-0.5;
3137                          SetCtrlVal( g_handle,     G_SETUP_Y_Min_G_4,      Y_Min_G_4);
3138                    //Read Graph #5
3139                    GetCtrlVal( g_handle,   G_SETUP_Var_G_5,        &plotVar_G_5);
3140                    //X
3141                        GetCtrlVal( g_handle,   G_SETUP_X_Mode_G_5,     &X_Mode_G_5);
3142                        GetCtrlVal( g_handle,   G_SETUP_X_Min_G_5,      &X_Min_G_5);
3143                        GetCtrlVal( g_handle,   G_SETUP_X_Max_G_5,      &X_Max_G_5);
3144                    if (X_Min_G_5 >= X_Max_G_5)
3145                          X_Min_G_5 = X_Max_G_5-0.5;
3146                          SetCtrlVal( g_handle,     G_SETUP_X_Min_G_5,      X_Min_G_5);
3147                    //Y
3148                        GetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_5,     &Y_Mode_G_5);
3149                        GetCtrlVal( g_handle,   G_SETUP_Y_Min_G_5,      &Y_Min_G_5);
3150                        GetCtrlVal( g_handle,   G_SETUP_Y_Max_G_5,      &Y_Max_G_5);
3151                    if (Y_Min_G_5 >= Y_Max_G_5)
3152                          Y_Min_G_5 = Y_Max_G_5-0.5;
3153                          SetCtrlVal( g_handle,     G_SETUP_Y_Min_G_5,      Y_Min_G_5);
3154
3155                    GetCtrlVal (panel_handle, MAIN_VIS_W,  &VIS_WATER);
3156                    GetCtrlVal (panel_handle, MAIN_VIS_O,  &VIS_OIL);
3157                    GetCtrlVal (panel_handle, MAIN_VIS_EM, &VIS_EMUL);
3158
3159                //Units for permeability and viscosity
3160                    vispermunit();
3161
3162                // COM Panel
3163                    GetConfigParms();
3164
3165                return;
3166                }
3167
3168                /*---------------------------------------------------------------------------*/
3169                /* Get the port configuration parameters.                                    */
3170                /*---------------------------------------------------------------------------*/
3171
3172                void GetConfigParms (void)
3173                {
3174                    GetCtrlVal (config_handle, CONFIG_CAMNAME, camName);
3175
3176                    // COM Panel
3177                    //#1
3178                    GetCtrlVal (config_handle, CONFIG_P1_COMPORT, &P1_comport);
3179                    GetCtrlVal (config_handle, CONFIG_P1_BAUDRATE, &P1_baudrate);
3180                    GetCtrlVal (config_handle, CONFIG_P1_PARITY, &P1_parity);
3181                    GetCtrlVal (config_handle, CONFIG_P1_DATABITS, &P1_databits);
3182                    GetCtrlVal (config_handle, CONFIG_P1_STOPBITS, &P1_stopbits);
3183                    GetCtrlVal (config_handle, CONFIG_P1_INPUTQ, &P1_inputq);
3184                    GetCtrlVal (config_handle, CONFIG_P1_OUTPUTQ, &P1_outputq);
3185                    GetCtrlIndex (config_handle, CONFIG_P1_COMPORT, &P1_portindex);
3186                    //#2
3187                    GetCtrlVal (config_handle, CONFIG_P2_COMPORT, &P2_comport);
3188                    GetCtrlVal (config_handle, CONFIG_P2_BAUDRATE, &P2_baudrate);
3189                    GetCtrlVal (config_handle, CONFIG_P2_PARITY, &P2_parity);
3190                    GetCtrlVal (config_handle, CONFIG_P2_DATABITS, &P2_databits);
3191                    GetCtrlVal (config_handle, CONFIG_P2_STOPBITS, &P2_stopbits);
3192                    GetCtrlVal (config_handle, CONFIG_P2_INPUTQ, &P2_inputq);
3193                    GetCtrlVal (config_handle, CONFIG_P2_OUTPUTQ, &P2_outputq);
3194                    GetCtrlIndex (config_handle, CONFIG_P2_COMPORT, &P2_portindex);
3195                    //#3
```

```
    GetCtrlVal (config_handle, CONFIG_P3_COMPORT, &P3_comport);
    GetCtrlVal (config_handle, CONFIG_P3_BAUDRATE, &P3_baudrate);
    GetCtrlVal (config_handle, CONFIG_P3_PARITY, &P3_parity);
    GetCtrlVal (config_handle, CONFIG_P3_DATABITS, &P3_databits);
    GetCtrlVal (config_handle, CONFIG_P3_STOPBITS, &P3_stopbits);
    GetCtrlVal (config_handle, CONFIG_P3_INPUTQ, &P3_inputq);
    GetCtrlVal (config_handle, CONFIG_P3_OUTPUTQ, &P3_outputq);
    GetCtrlIndex (config_handle, CONFIG_P3_COMPORT, &P3_portindex);
}

//*********************************************************
// Load_Vars    : Load variables to the screen
// build based on DSN_Load_Vars
//*********************************************************

void LoadVars(void)
{
//MAIN PANEL
    // Main Parameters
    SetCtrlVal(panel_handle, MAIN_LOG_Graph,        Is_Graph);
    SetCtrlVal(panel_handle, MAIN_LOG_LOGtoFile,    Is_Log);
    // COM Ports (data from pumps)
    SetCtrlVal(panel_handle, MAIN_LOG_P1,           Is_P1);
    SetCtrlVal(panel_handle, MAIN_LOG_P2,           Is_P2);
    SetCtrlVal(panel_handle, MAIN_LOG_P3,           Is_P3);
    // Mass Flow Meters
    SetCtrlVal(panel_handle, MAIN_LOG_M1,           Is_M1);
    SetCtrlVal(panel_handle, MAIN_LOG_M2,           Is_M2);
    SetCtrlVal(panel_handle, MAIN_LOG_M3,           Is_M3);
    SetCtrlVal(panel_handle, MAIN_LOG_M4,           Is_M4);
    // Back Pressure Regulator
    SetCtrlVal(panel_handle, MAIN_LOG_BPR,          Is_BPR);
    // DAQ
    SetCtrlVal(panel_handle, MAIN_LOG_DAQ,          Is_DAQ);

// GRAPH PANEL
    //Read Graph #1
    SetCtrlVal( g_handle,   G_SETUP_Var_G_1,        plotVar_G_1);
    SetCtrlVal( g_handle,   G_SETUP_X_Range_G_1,    X_Range_G_1);
    SetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_1,     Y_Mode_G_1);
    SetCtrlVal( g_handle,   G_SETUP_Y_Min_G_1,      Y_Min_G_1);
    SetCtrlVal( g_handle,   G_SETUP_Y_Max_G_1,      Y_Max_G_1);
    //Read Graph #2
    SetCtrlVal( g_handle,   G_SETUP_Var_G_2,        plotVar_G_2);
    SetCtrlVal( g_handle,   G_SETUP_X_Range_G_2,    X_Range_G_2);
    SetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_2,     Y_Mode_G_2);
    SetCtrlVal( g_handle,   G_SETUP_Y_Min_G_2,      Y_Min_G_2);
    SetCtrlVal( g_handle,   G_SETUP_Y_Max_G_2,      Y_Max_G_2);
    //Read Graph #3
    SetCtrlVal( g_handle,   G_SETUP_Var_G_3,        plotVar_G_3);
    SetCtrlVal( g_handle,   G_SETUP_X_Range_G_3,    X_Range_G_3);
    SetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_3,     Y_Mode_G_3);
    SetCtrlVal( g_handle,   G_SETUP_Y_Min_G_3,      Y_Min_G_3);
    SetCtrlVal( g_handle,   G_SETUP_Y_Max_G_3,      Y_Max_G_3);
    //Read Graph #4
    SetCtrlVal( g_handle,   G_SETUP_Var_G_4,        plotVar_G_4);
    SetCtrlVal( g_handle,   G_SETUP_X_Range_G_4,    X_Range_G_4);
    SetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_4,     Y_Mode_G_4);
    SetCtrlVal( g_handle,   G_SETUP_Y_Min_G_4,      Y_Min_G_4);
    SetCtrlVal( g_handle,   G_SETUP_Y_Max_G_4,      Y_Max_G_4);

    //Read Graph #5
    SetCtrlVal( g_handle,   G_SETUP_Var_G_5,        plotVar_G_5);

    SetCtrlVal( g_handle,   G_SETUP_X_Mode_G_5,     X_Mode_G_5);
    SetCtrlVal( g_handle,   G_SETUP_X_Min_G_5,      X_Min_G_5);
    SetCtrlVal( g_handle,   G_SETUP_X_Max_G_5,      X_Max_G_5);

    SetCtrlVal( g_handle,   G_SETUP_Y_Mode_G_5,     Y_Mode_G_5);
    SetCtrlVal( g_handle,   G_SETUP_Y_Min_G_5,      Y_Min_G_5);
    SetCtrlVal( g_handle,   G_SETUP_Y_Max_G_5,      Y_Max_G_5);

    SetCtrlVal (panel_handle, MAIN_WATERTANK, G_WATERINJ[0]);
    SetCtrlVal (panel_handle, MAIN_OILTANK,  G_OILINJ[0]);
    SetCtrlVal (panel_handle, MAIN_EMULTANK, G_EMULINJ[0]);

    SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);

    SetCtrlVal (panel_handle, MAIN_VIS_W, VIS_WATER);
    SetCtrlVal (panel_handle, MAIN_VIS_O,  VIS_OIL);
    SetCtrlVal (panel_handle, MAIN_VIS_EM, VIS_EMUL);

    if (viscoef == 1)  SetCtrlVal (panel_handle, MAIN_VIS_UNIT, 2);
        else  SetCtrlVal (panel_handle, MAIN_VIS_UNIT, 0);
    if (permcoef == 1) SetCtrlVal (panel_handle, MAIN_PERM_UNIT, 0);
        else  SetCtrlVal (panel_handle, MAIN_PERM_UNIT, 1);

    G_WATERINJ[1]=G_WATERINJ[0];
    G_OILINJ[1]= G_OILINJ[0];
    G_EMULINJ[1]=G_EMULINJ[0];

    OUTLET[1]=OUTLET[0];


// COM PANEL
    SetConfigParms ();
```

```
3293                return;
3294                }
3295
3296                    /*---------------------------------------------------------------------------*/
3297                    /* Set configuration parameters.                                             */
3298                    /*---------------------------------------------------------------------------*/
3299
3300                    void SetConfigParms (void)
3301                    {
3302                        // Camera
3303                        SetCtrlVal (config_handle, CONFIG_CAMNAME,    camName);
3304
3305                        // COM Panel
3306                        //#1
3307                        SetCtrlVal (config_handle, CONFIG_P1_COMPORT,  P1_comport);
3308                        SetCtrlVal (config_handle, CONFIG_P1_BAUDRATE, P1_baudrate);
3309                        SetCtrlVal (config_handle, CONFIG_P1_PARITY,   P1_parity);
3310                        SetCtrlVal (config_handle, CONFIG_P1_DATABITS, P1_databits);
3311                        SetCtrlVal (config_handle, CONFIG_P1_STOPBITS, P1_stopbits);
3312                        SetCtrlVal (config_handle, CONFIG_P1_INPUTQ,   P1_inputq);
3313                        SetCtrlVal (config_handle, CONFIG_P1_OUTPUTQ,  P1_outputq);
3314                        SetCtrlVal (config_handle, CONFIG_P1_CTSMODE,  P1_ctsmode);
3315                        SetCtrlVal (config_handle, CONFIG_P1_XMODE,    P1_xmode);
3316                        SetCtrlVal (config_handle, CONFIG_P1_TIMEOUT,  P1_timeout);
3317                        SetCtrlIndex (config_handle, CONFIG_P1_COMPORT,P1_portindex);
3318                        //#2
3319                        SetCtrlVal (config_handle, CONFIG_P2_COMPORT,  P2_comport);
3320                        SetCtrlVal (config_handle, CONFIG_P2_BAUDRATE, P2_baudrate);
3321                        SetCtrlVal (config_handle, CONFIG_P2_PARITY,   P2_parity);
3322                        SetCtrlVal (config_handle, CONFIG_P2_DATABITS, P2_databits);
3323                        SetCtrlVal (config_handle, CONFIG_P2_STOPBITS, P2_stopbits);
3324                        SetCtrlVal (config_handle, CONFIG_P2_INPUTQ,   P2_inputq);
3325                        SetCtrlVal (config_handle, CONFIG_P2_OUTPUTQ,  P2_outputq);
3326                        SetCtrlVal (config_handle, CONFIG_P2_CTSMODE,  P2_ctsmode);
3327                        SetCtrlVal (config_handle, CONFIG_P2_XMODE,    P2_xmode);
3328                        SetCtrlVal (config_handle, CONFIG_P2_TIMEOUT,  P2_timeout);
3329                        SetCtrlIndex (config_handle, CONFIG_P2_COMPORT,P2_portindex);
3330                        //#3
3331                        SetCtrlVal (config_handle, CONFIG_P3_COMPORT,  P3_comport);
3332                        SetCtrlVal (config_handle, CONFIG_P3_BAUDRATE, P3_baudrate);
3333                        SetCtrlVal (config_handle, CONFIG_P3_PARITY,   P3_parity);
3334                        SetCtrlVal (config_handle, CONFIG_P3_DATABITS, P3_databits);
3335                        SetCtrlVal (config_handle, CONFIG_P3_STOPBITS, P3_stopbits);
3336                        SetCtrlVal (config_handle, CONFIG_P3_INPUTQ,   P3_inputq);
3337                        SetCtrlVal (config_handle, CONFIG_P3_OUTPUTQ,  P3_outputq);
3338                        SetCtrlVal (config_handle, CONFIG_P3_CTSMODE,  P3_ctsmode);
3339                        SetCtrlVal (config_handle, CONFIG_P3_XMODE,    P3_xmode);
3340                        SetCtrlVal (config_handle, CONFIG_P3_TIMEOUT,  P3_timeout);
3341                        SetCtrlIndex (config_handle, CONFIG_P3_COMPORT,P3_portindex);
3342                    }
3343
3344 //************************************************************
3345 // OnOff_Graphs     : For Turning Graphs ON and OFF (Dimming)
3346 // same as DSN_OnOff_Graphs
3347 //************************************************************
3348
3349 int CVICALLBACK OnOff_Graphs (int panel, int control, int event,
3350         void *callbackData, int eventData1, int eventData2)
3351 {   int test;
3352     switch (event)
3353     {
3354         case EVENT_COMMIT:
3355             switch(control)
3356             {
3357                                 case G_SETUP_OnOff_G_1:
3358                                     GetCtrlVal (g_handle, G_SETUP_OnOff_G_1, &test);
3359                                     if(test)
3360                                       {
3361                                       SetCtrlAttribute (panel_handle, MAIN_G_1, ATTR_VISIBLE, 1);
3362                                       OnOff_G1 = 1;
3363                                       }
3364                                     else
3365                                       {
3366                                       SetCtrlAttribute (panel_handle, MAIN_G_1, ATTR_VISIBLE, 0);
3367                                       OnOff_G1 = 0;
3368                                       }
3369                                     break;
3370                                 case G_SETUP_OnOff_G_2:
3371                                     GetCtrlVal (g_handle, G_SETUP_OnOff_G_2, &test);
3372                                     if(test)
3373                                       {
3374                                       SetCtrlAttribute (panel_handle, MAIN_G_2, ATTR_VISIBLE, 1);
3375                                       OnOff_G2 = 1;
3376                                       }
3377                                     else
3378                                       {
3379                                       SetCtrlAttribute (panel_handle, MAIN_G_2, ATTR_VISIBLE, 0);
3380                                       OnOff_G2 = 0;
3381                                       }
3382                                     break;
3383                                 case G_SETUP_OnOff_G_3:
3384                                     GetCtrlVal (g_handle, G_SETUP_OnOff_G_3, &test);
3385                                     if(test)
3386                                       {
3387                                       SetCtrlAttribute (panel_handle, MAIN_G_3, ATTR_VISIBLE, 1);
3388                                       OnOff_G3 = 1;
3389                                       }
```

```
                                    else
                                      {
                                      SetCtrlAttribute (panel_handle, MAIN_G_3, ATTR_VISIBLE, 0);
                                      OnOff_G3 = 0;
                                      }
                                    break;
                                case G_SETUP_OnOff_G_4:
                                    GetCtrlVal (g_handle, G_SETUP_OnOff_G_4, &test);
                                    if(test)
                                      {
                                      SetCtrlAttribute (panel_handle, MAIN_G_4, ATTR_VISIBLE, 1);
                                      OnOff_G4 = 1;
                                      }
                                    else
                                      {
                                      SetCtrlAttribute (panel_handle, MAIN_G_4, ATTR_VISIBLE, 0);
                                      OnOff_G4 = 0;
                                      }
                                    break;
                                case G_SETUP_OnOff_G_5:
                                    GetCtrlVal (g_handle, G_SETUP_OnOff_G_5, &test);
                                    if(test)
                                      {
                                      DisplayPanel(extra_g_handle);
                                      OnOff_G5 = 1;
                                      }
                                    else
                                      {
                                      HidePanel (extra_g_handle);
                                      OnOff_G5 = 0;
                                      }
                                    break;

                                case G_SETUP_OnOff_G_6:
                                    GetCtrlVal (g_handle, G_SETUP_OnOff_G_6, &test);
                                    if(test)
                                      {
                                      OnOff_G6 = 1;
                                      }
                                    else
                                      {
                                      OnOff_G6 = 0;
                                      }
                                    break;
            }
            break;
        case EVENT_RIGHT_CLICK:
            break;
    }
    return 0;
}

int CVICALLBACK SHOW_Graphs (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            break;
        case EVENT_RIGHT_CLICK:
                DisplayPanel(g_handle);

            break;

    }
    return 0;
}

/*****************************************************************/
/* Pumps_ThreadFunction ():  - Separate Thread for Pumps         */
/*****************************************************************/

int CVICALLBACK Pumps_ThreadFunction (void *functionData)   //connecting to the pumps
{

while (Pumps_quitflag == 1)
      {
      ProcessSystemEvents ();
      Delay (0.5);
        if (Is_P1) { CheckPressurePump1(); }  //update information about pressure
      Delay (0.5);
        if (Is_P2) { CheckPressurePump2(); }  //update information about pressure
      Delay (0.5);
        if (Is_P3) { CheckPressurePump3(); } //update information about pressure

      ProcessSystemEvents ();
      }
return 0;
}


int CVICALLBACK Camera_ThreadFunction(void *functionData)
{
while (Camera_quitflag == 1)
      {
      ProcessSystemEvents ();
```

```
3487        Delay (0.5);
3488          if (Is_Cam)
3489          {   // RUN the timer to do the GRAB and DISPLAY
3490            SetCtrlAttribute (panel_handle, MAIN_TIMER_2, ATTR_ENABLED, TRUE);  }
3491          else                // RUN the timer to do the GRAB and DISPLAY
3492                  SetCtrlAttribute (panel_handle, MAIN_TIMER_2, ATTR_ENABLED, FALSE);
3493          //update information about pressure
3494        ProcessSystemEvents ();
3495        }
3496 return 0;
3497 }
3498
3499 int CVICALLBACK Email_ThreadFunction(void *functionData)
3500 {
3501 double temptube;
3502 while (Email_quitflag == 1)
3503        {
3504        ProcessSystemEvents ();
3505        Delay (0.5);
3506
3507          if (globalemail == 1)
3508          {
3509                  GetCtrlVal(panel_handle, MAIN_CUR_TTUBE, &temptube);
3510
3511              //Generate results
3512                 sprintf(currentresuts, "cmd.exe /c sendEmail -f cmcprojectlab@gmail.com -t 3666377@gmail.com -s smtp.gmail.com:587
      -xu cmcprojectlab@gmail.com  -xp cmcproject2013 -u \"cmcproject\" -m \"TEST TUBE = %3.1f WATER INJ = %4.2f EMUL INJ = %4.2f This
      message has been sent by software\"", temptube,WATERINJ2PV[1],EMULIN[1]);
3513              //Send email with results
3514                 system(currentresuts);
3515              globalemail = 0;
3516          }
3517
3518        ProcessSystemEvents ();
3519        }
3520 return 0;
3521 }
3522
3523
3524
3525
3526 int CVICALLBACK Mass_ThreadFunction (void *functionData)    //connecting to the mass flow meters
3527 { int conreq_M1,
3528        conreq_M2,
3529        conreq_M3,
3530        conreq_M4;
3531
3532 while (Mass_quitflag == 1)
3533        {
3534        ProcessSystemEvents ();
3535
3536
3537        GetCtrlVal(config_handle,CONFIG_CON_M_1,&conreq_M1);
3538        GetCtrlVal(config_handle,CONFIG_CON_M_2,&conreq_M2);
3539        GetCtrlVal(config_handle,CONFIG_CON_M_3,&conreq_M3);
3540        GetCtrlVal(config_handle,CONFIG_CON_M_4,&conreq_M4);
3541
3542            //Mass flow meter #1 Water//
3543              if(conreq_M1 == 1)
3544              {
3545                     ConnectToDDEServer (&hConv, "FlowDDE", "C(1)", DDECallback, 0);
3546                     conreq_M1=0; Is_M1=1;
3547                     SetCtrlVal(config_handle,CONFIG_CON_M_1, conreq_M1);
3548                     SetCtrlVal(panel_handle,MAIN_LOG_M1, Is_M1);
3549                     SetCtrlVal (config_handle, CONFIG_M1_LED, 1);
3550                     Update_Graphics();
3551              }
3552              else if (conreq_M1 == -1)
3553              {
3554                     DisconnectFromDDEServer (hConv);
3555                     conreq_M1=0; Is_M1=0;
3556                     SetCtrlVal(config_handle,CONFIG_CON_M_1, conreq_M1);
3557                     SetCtrlVal(panel_handle,MAIN_LOG_M1, Is_M1);
3558                     SetCtrlVal (config_handle, CONFIG_M1_LED, 0);
3559                     Update_Graphics();
3560              }
3561                  if (Is_M1) READ_FLOW_M_1();
3562            //Mass flow meter #1 Water//
3563
3564               //Mass flow meter #2 Oil//
3565                 if(conreq_M2 == 1)
3566                 {
3567                        ConnectToDDEServer (&hConv2, "FlowDDE2", "C(1)", DDECallback, 0);
3568                        conreq_M2=0; Is_M2=1;
3569                        SetCtrlVal(config_handle,CONFIG_CON_M_2, conreq_M2);
3570                        SetCtrlVal(panel_handle,MAIN_LOG_M2, Is_M2);
3571                        SetCtrlVal (config_handle, CONFIG_M2_LED, 1);
3572                        Update_Graphics();
3573                 }
3574                 else if (conreq_M2 == -1)
3575                 {
3576                        DisconnectFromDDEServer (hConv2);
3577                        conreq_M2=0; Is_M2=0;
3578                        SetCtrlVal(config_handle,CONFIG_CON_M_2, conreq_M2);
3579                        SetCtrlVal(panel_handle,MAIN_LOG_M2, Is_M2);
3580                        SetCtrlVal (config_handle, CONFIG_M2_LED, 0);
3581                        Update_Graphics();
```

```
3582                }
3583                    if (Is_M2) READ_FLOW_M_2();
3584             //Mass flow meter #2 Oil//
3585
3586                     //Mass flow meter #3 Emulsion//
3587                       if(conreq_M3 == 1)
3588                       {
3589                              ConnectToDDEServer (&hConv3, "FlowDDE3", "C(1)", DDECallback, 0);
3590                              conreq_M3=0; Is_M3=1;
3591                              SetCtrlVal(config_handle,CONFIG_CON_M_3, conreq_M3);
3592                              SetCtrlVal(panel_handle,MAIN_LOG_M3, Is_M3);
3593                              SetCtrlVal (config_handle, CONFIG_M3_LED, 1);
3594                              Update_Graphics();
3595                       }
3596                       else if (conreq_M3 == -1)
3597                       {
3598                              DisconnectFromDDEServer (hConv3);
3599                              conreq_M3=0; Is_M3=0;
3600                              SetCtrlVal(config_handle,CONFIG_CON_M_3, conreq_M3);
3601                              SetCtrlVal(panel_handle,MAIN_LOG_M3, Is_M3);
3602                              SetCtrlVal (config_handle, CONFIG_M3_LED, 0);
3603                              Update_Graphics();
3604                       }
3605                              if (Is_M3) READ_FLOW_M_3();
3606                     //Mass flow meter #3 Emulsion//
3607
3608                            //Mass flow meter #4 Mixture//
3609                              if(conreq_M4 == 1)
3610                              {
3611                                     ConnectToDDEServer (&hConv4, "FlowDDE4", "C(1)", DDECallback, 0);
3612                                     conreq_M4=0; Is_M4=1;
3613                                     SetCtrlVal(config_handle,CONFIG_CON_M_4, conreq_M4);
3614                                     SetCtrlVal(panel_handle,MAIN_LOG_M4, Is_M4);
3615                                     SetCtrlVal (config_handle, CONFIG_M4_LED, 1);
3616                                     Update_Graphics();
3617                              }
3618                              else if (conreq_M4 == -1)
3619                              {
3620                                     DisconnectFromDDEServer (hConv4);
3621                                     conreq_M4=0; Is_M4=0;
3622                                     SetCtrlVal(config_handle,CONFIG_CON_M_4, conreq_M4);
3623                                     SetCtrlVal(panel_handle,MAIN_LOG_M4, Is_M4);
3624                                     SetCtrlVal (config_handle, CONFIG_M4_LED, 0);
3625                                     Update_Graphics();
3626                              }
3627                                  if (Is_M4) READ_FLOW_M_4();
3628                            //Mass flow meter #4 Mixture//
3629
3630      ProcessSystemEvents ();
3631      }
3632 return 0;
3633 }
3634
3635 void READ_FLOW_M_1(void)
3636 {
3637 double
3638        data,
3639        data2,
3640        data3,
3641        data4;
3642
3643 ProcessSystemEvents ();
3644
3645                 Delay(0.01);
3646                 ClientDDERead (hConv, "P(270)", CF_TEXT, read_m11data, 64, 0);
3647                 data=atof(read_m11data);
3648                 SetCtrlVal (panel_handle, MAIN_DENSITY_M_1, data);  // send value of "data" to panel
3649
3650                 Delay(0.01);
3651                 ClientDDERead (hConv, "P(142)", CF_TEXT, read_m12data, 64, 0);
3652                 data2=atof(read_m12data);
3653                 SetCtrlVal (panel_handle, MAIN_TEMP_M_1, data2);
3654
3655                 Delay(0.01);
3656                 ClientDDERead (hConv, "P(205)", CF_TEXT, read_m13data, 64, 0);
3657                 data3=atof(read_m13data);
3658                 SetCtrlVal (panel_handle, MAIN_MFR_M_1, data3);
3659
3660                 data4=data3 * data * 0.001 ;       // cc/min * kg /m^3 = 0.001 * gm/min
3661
3662             //  Delay(0.01);           //P(198)
3663             //  ClientDDERead (hConv, "P(205)", CF_TEXT, read_m14data, 64, 0);
3664             //  data4=atof(read_m14data);
3665
3666                 SetCtrlVal (panel_handle, MAIN_EXTRA1_M_1, data4);
3667
3668     CmtGetLock (Mass_lockHandle);
3669        DSN_SHIFT_MASS(0);
3670        M11[0] = data;
3671        M12[0] = data2;
3672        M13[0] = data3;   //volume flow rate cc/min _ P205
3673        M14[0] = data4;   //Real mass flow in kg/min
3674     CmtReleaseLock (Mass_lockHandle);
3675
3676
3677 ProcessSystemEvents ();
3678 }
```

```
3679
3680 void READ_FLOW_M_2(void)
3681 {
3682 double data,
3683        data2,
3684        data3,
3685        data4;
3686
3687 ProcessSystemEvents ();
3688
3689                 Delay(0.01);
3690                 ClientDDERead (hConv2, "P(270)", CF_TEXT, read_m21data, 64, 0);
3691                 data=atof(read_m21data);
3692                 SetCtrlVal (panel_handle, MAIN_DENSITY_M_2, data);  // send value of "data" to panel
3693
3694                 Delay(0.01);
3695                 ClientDDERead (hConv2, "P(142)", CF_TEXT, read_m22data, 64, 0);
3696                 data2=atof(read_m22data);
3697                 SetCtrlVal (panel_handle, MAIN_TEMP_M_2, data2);
3698
3699                 Delay(0.01);
3700                 ClientDDERead (hConv2, "P(205)", CF_TEXT, read_m23data, 64, 0);
3701                 data3=atof(read_m23data);
3702                 SetCtrlVal (panel_handle, MAIN_MFR_M_2, data3);
3703
3704                 data4=data3 * data * 0.001 ;       // cc/min * kg /m^3 = 0.001 * gm/min
3705
3706             //  Delay(0.01);            //P(198)
3707             //  ClientDDERead (hConv2, "P(205)", CF_TEXT, read_m24data, 64, 0);
3708             //  data4=atof(read_m24data);
3709
3710                 SetCtrlVal (panel_handle, MAIN_EXTRA1_M_2, data4);
3711
3712
3713     CmtGetLock (Mass_lockHandle);
3714        DSN_SHIFT_MASS(1);
3715        M21[0] = data;
3716        M22[0] = data2;
3717        M23[0] = data3;  //volume flow rate in cc/min
3718        M24[0] = data4;  //real mass flow in kg/min
3719     CmtReleaseLock (Mass_lockHandle);
3720
3721
3722
3723
3724 ProcessSystemEvents ();
3725 }
3726
3727 void READ_FLOW_M_3(void)
3728 {
3729 double data,
3730        data2,
3731        data3,
3732        data4,
3733        temp_time_e=0.0,
3734        temp_delta_e=0.0,
3735        temp_emul_in=0.0,
3736        temp_emul_in_pv=0.0;
3737
3738 ProcessSystemEvents ();
3739
3740                 Delay(0.01);
3741                 ClientDDERead (hConv3, "P(270)", CF_TEXT, read_m31data, 64, 0);
3742                 data=atof(read_m31data);
3743                 SetCtrlVal (panel_handle, MAIN_DENSITY_M_3, data);  // send value of "data" to panel
3744
3745                 Delay(0.01);
3746                 ClientDDERead (hConv3, "P(142)", CF_TEXT, read_m32data, 64, 0);
3747                 data2=atof(read_m32data);
3748                 SetCtrlVal (panel_handle, MAIN_TEMP_M_3, data2);
3749
3750                 Delay(0.01);
3751                 ClientDDERead (hConv3, "P(205)", CF_TEXT, read_m33data, 64, 0);
3752                 data3=atof(read_m33data);
3753                 SetCtrlVal (panel_handle, MAIN_MFR_M_3, data3);
3754
3755                 data4=data3 * data * 0.001 ;       // cc/min * kg /m^3 = 0.001 * gm/min
3756
3757             //  Delay(0.01);            //P(198)
3758             //  ClientDDERead (hConv3, "P(205)", CF_TEXT, read_m34data, 64, 0);
3759             //  data4=atof(read_m34data);
3760
3761                 SetCtrlVal (panel_handle, MAIN_EXTRA1_M_3, data4);
3762
3763 Gtime_E [0] = Timer ();  // Get Time
3764
3765 if (stage ==10 && pumprun_3 == 31)     //emulsion flooding
3766 {
3767 temp_time_e = Gtime_E [0] - Gtime_E [1];    //delta time
3768 temp_delta_e=data3*temp_time_e/60;      //delta mass
3769
3770 temp_emul_in = EMULIN[0] + temp_delta_e;  //emulsion inj based on mass flow meter
3771
3772 temp_emul_in_pv = temp_emul_in / PV[0];
3773
3774 SetCtrlVal (panel_handle, MAIN_EMUL_IN, temp_emul_in);
3775 SetCtrlVal (panel_handle, MAIN_EMUL_IN_PV, temp_emul_in_pv);
```

```
3776
3777 }
3778
3779         if (stage ==11 && pumprun_3 == 31)     //alternate emulsion flooding
3780         {
3781         temp_time_e = Gtime_E [0] - Gtime_E [1];    //delta time
3782         temp_delta_e=data3*temp_time_e/60;      //delta mass
3783
3784         temp_emul_in = EMULIN[0] + temp_delta_e;  //emulsion inj based on mass flow meter
3785
3786         temp_emul_in_pv = temp_emul_in / PV[0];
3787
3788         SetCtrlVal (panel_handle, MAIN_E_ALT_INJ, temp_emul_in);
3789         SetCtrlVal (panel_handle, MAIN_EMULSION_PV_INJ_ALT, temp_emul_in_pv);
3790         }
3791
3792
3793
3794
3795     CmtGetLock (Mass_lockHandle);
3796        DSN_SHIFT_MASS(2);
3797        M31[0] = data;
3798        M32[0] = data2;
3799        M33[0] = data3;  //volume flow rate cc/min
3800        M34[0] = data4;  //real mass flow in kg/min
3801
3802                         if (stage >=10 && pumprun_3 == 31)     //emulsion flooding or alternate emulsion flooding
3803                         {
3804                          TEMPSTEP_E[0] = temp_time_e;
3805                          TEMPDELTA_E[0]= temp_delta_e;
3806                          EMULIN[0] =  temp_emul_in;
3807                         }
3808
3809     CmtReleaseLock (Mass_lockHandle);
3810
3811 ProcessSystemEvents ();
3812 }
3813
3814 void READ_FLOW_M_4(void)
3815 {
3816 double data,
3817        data2,
3818        data3,
3819        data4,
3820        temp_delta=0.0,
3821        temp_time=0.0,
3822        temp_g_time=0.0,
3823        temp_waterflooding_inj = 0.0,
3824        temp_sior = 0.0 ,
3825        temp_water_rec=0.0,          //waterflooding
3826        temp_oil_rec=0.0,            //waterflooding
3827        temp_water_rec_st9=0.0,      //oil saturation
3828        temp_oil_rec_st9=0.0,        //oil saturation
3829        temp_swi = 0.0,
3830        temp_oil_rec_st14 = 0.0,     //emulsion flooding
3831        temp_smth_rec_st14 = 0.0,    //emulsion flooding
3832        temp_smth_rec_ALT = 0.0,     //alternate flooding
3833        temp_oil_rec_ALT = 0.0;      //alternate flooding
3834
3835 ProcessSystemEvents ();
3836
3837                 Delay(0.01);
3838                 ClientDDERead (hConv4, "P(270)", CF_TEXT, read_m41data, 64, 0);
3839                 data=atof(read_m41data);
3840                 SetCtrlVal (panel_handle, MAIN_DENSITY_M_4, data);  // send value of "read_data" to panel
3841
3842                 Delay(0.01);
3843                 ClientDDERead (hConv4, "P(142)", CF_TEXT, read_m42data, 64, 0);
3844                 data2=atof(read_m42data);
3845                 SetCtrlVal (panel_handle, MAIN_TEMP_M_4, data2);
3846
3847                 Delay(0.01);
3848                 ClientDDERead (hConv4, "P(205)", CF_TEXT, read_m43data, 64, 0);
3849                 data3=atof(read_m43data);
3850                 SetCtrlVal (panel_handle, MAIN_MFR_M_4, data3);
3851
3852                 data4=data3 * data * 0.001 ;       // cc/min * kg /m^3 = 0.001 * gm/min
3853
3854             //  Delay(0.01);            //P(198)
3855             //  ClientDDERead (hConv4, "P(205)", CF_TEXT, read_m44data, 64, 0);
3856             //  data4=atof(read_m44data);
3857
3858                 SetCtrlVal (panel_handle, MAIN_EXTRA1_M_4, data4);
3859
3860 Gtime[0] = Timer ();  // Get Time
3861
3862 if (stage == 4 && pumprun_2 == 21)   //oil saturation process [stage #4]
3863 {
3864
3865 temp_time = Gtime [0] - Gtime [1];
3866
3867 temp_delta=data3*temp_time/60;
3868
3869      if (M41[0] >= SETPOINTOUT[0])
3870      {
3871          temp_water_rec_st9 = WATERREC_ST9[0]+ temp_delta;
3872          temp_oil_rec_st9 = OILREC_ST9[0];
```

```
3873      }
3874      else
3875      {
3876          temp_oil_rec_st9 =  OILREC_ST9[0] + temp_delta;
3877          temp_water_rec_st9 = WATERREC_ST9[0];
3878      }
3879
3880
3881 temp_swi =  (PV[0] -  temp_water_rec_st9 + 0.46) / PV[0];   //0.46 - shift-volume before and after CORE HOLDER
3882
3883 SetCtrlVal (panel_handle, MAIN_SIW_CORE, temp_swi);
3884
3885 SetCtrlVal (panel_handle, MAIN_WATER_REC, temp_water_rec_st9);
3886 SetCtrlVal (panel_handle, MAIN_OIL_REC, temp_oil_rec_st9);
3887 }
3888
3889 if (stage == 7 && pumprun_1 == 11 )    // native waterflooding
3890 {
3891 temp_time = Gtime [0] - Gtime [1];
3892 temp_delta=data3*temp_time/60;
3893
3894      if (M41[0] >= SETPOINTOUT[0])
3895      {
3896          temp_water_rec = WATERREC[0]+ temp_delta;
3897          temp_oil_rec = OILREC[0];
3898      }
3899      else
3900      {
3901          temp_oil_rec =  OILREC[0] + temp_delta;
3902          temp_water_rec = WATERREC [0];
3903      }
3904
3905 temp_waterflooding_inj = WATERFLOODINJ [0] + PFR11[0] *temp_time/60;  // water inject during waterflooding
3906
3907 // it is required to take into account water injected [temp_waterflooding_inj]
3908
3909 temp_swi= (PV[0] - WATERREC_ST9[0] - temp_water_rec + temp_waterflooding_inj + 0.46) / PV[0];
3910 temp_sior = 1 - temp_swi ;
3911
3912
3913 SetCtrlVal (panel_handle, MAIN_SIW_CORE, temp_swi);
3914 SetCtrlVal (panel_handle, MAIN_SIOR_CORE,temp_sior);
3915
3916
3917 SetCtrlVal (panel_handle, MAIN_WATER_REC, temp_water_rec);
3918 SetCtrlVal (panel_handle, MAIN_OIL_REC, temp_oil_rec);
3919 }
3920
3921 if (stage == 10 && pumprun_3 == 31)     //emulsion flooding
3922 {
3923 temp_time = Gtime [0] - Gtime [1];
3924 temp_delta=data3*temp_time/60;
3925
3926       if (M41[0] >= SETPOINTOUT[0])
3927       {
3928         temp_smth_rec_st14 = RECEMUL[0] + temp_delta;
3929         temp_oil_rec_st14 = OILREC_ST14 [0];
3930       }
3931       else
3932       {
3933          temp_oil_rec_st14 =  OILREC_ST14 [0] + temp_delta;
3934          temp_smth_rec_st14 = RECEMUL [0];
3935       }
3936
3937
3938 SetCtrlVal (panel_handle, MAIN_WATER_REC_2, temp_smth_rec_st14);
3939 SetCtrlVal (panel_handle, MAIN_OIL_REC_2,  temp_oil_rec_st14);
3940 }
3941
3942 if (stage == 11 && pumprun_1 == 11) //alternate flooding  [waterflooding]
3943 {
3944 SetCtrlVal (panel_handle, MAIN_ALT_P_1, 1);  //on
3945 SetCtrlVal (panel_handle, MAIN_ALT_P_3, 0);  //off
3946
3947 temp_time = Gtime [0] - Gtime [1];
3948 temp_delta=data3*temp_time/60;
3949
3950       if (M41[0] >= SETPOINTOUT[0])
3951       {
3952         temp_smth_rec_ALT = RECEMUL_ALT[0] + temp_delta;
3953         temp_oil_rec_ALT =  OILREC_ALT[0];
3954       }
3955       else
3956       {
3957          temp_oil_rec_ALT =  OILREC_ALT[0] + temp_delta;
3958          temp_smth_rec_ALT = RECEMUL_ALT[0];
3959       }
3960
3961 SetCtrlVal (panel_handle, MAIN_ALT_W_REC, temp_smth_rec_ALT);
3962 SetCtrlVal (panel_handle, MAIN_ALT_O_REC, temp_oil_rec_ALT);
3963 }
3964
3965 if (stage == 11 && pumprun_3 == 31) //alternate flooding  [emulsion flooding]
3966 {
3967 SetCtrlVal (panel_handle, MAIN_ALT_P_1, 0); //off
3968 SetCtrlVal (panel_handle, MAIN_ALT_P_3, 1); //on
3969
```

```
3970 temp_time = Gtime [0] - Gtime [1];
3971 temp_delta=data3*temp_time/60;
3972
3973       if (M41[0] >= SETPOINTOUT[0])
3974       {
3975         temp_smth_rec_ALT = RECEMUL_ALT[0] + temp_delta;
3976         temp_oil_rec_ALT =  OILREC_ALT [0];
3977       }
3978       else
3979       {
3980         temp_oil_rec_ALT =  OILREC_ALT [0] + temp_delta;
3981         temp_smth_rec_ALT = RECEMUL_ALT [0];
3982       }
3983
3984 SetCtrlVal (panel_handle, MAIN_ALT_W_REC, temp_smth_rec_ALT);
3985 SetCtrlVal (panel_handle, MAIN_ALT_O_REC, temp_oil_rec_ALT);
3986 }
3987
3988 SetCtrlVal (panel_handle, MAIN_TIME_REAL, TimeStr());
3989
3990 //Save data to the massive elements
3991 CmtGetLock (Mass_lockHandle);
3992
3993        DSN_SHIFT_MASS(3);
3994
3995        M41[0] = data;
3996        M42[0] = data2;
3997        M43[0] = data3;   //volume flow rate in cc/min
3998        M44[0] = data4;   //real mass flow in kg/min
3999
4000                 if (stage == 4 && pumprun_2 == 21)      //oil saturation process
4001                 {
4002                      TEMPSTEP[0] = temp_time;
4003                      TEMPDELTA[0]= temp_delta;
4004                      WATERREC_ST9[0] = temp_water_rec_st9;
4005                      OILREC_ST9[0]   = temp_oil_rec_st9;
4006                      SWI[0] = temp_swi;
4007                 }
4008
4009                     if (stage == 7 && pumprun_1 == 11)  //native waterflooding
4010                     {
4011                           WATERREC[0] = temp_water_rec;
4012                           OILREC[0]   = temp_oil_rec;
4013                           TEMPSTEP[0] = temp_time;
4014                           TEMPDELTA[0]= temp_delta;
4015                           WATERFLOODINJ[0]= temp_waterflooding_inj;
4016                           SWI[0] = temp_swi;
4017                     }
4018
4019                          if (stage == 10 && pumprun_3 == 31)  //emulsion flooding
4020                          {
4021                           TEMPSTEP[0] = temp_time;
4022                           TEMPDELTA[0]= temp_delta;
4023                           OILREC_ST14 [0] =  temp_oil_rec_st14;
4024                           RECEMUL[0] =  temp_smth_rec_st14;
4025                          }
4026
4027                              if (stage == 11 && pumprun_1 == 11)     //alternate flooding [water]
4028                              {
4029                               TEMPSTEP[0] = temp_time;
4030                               TEMPDELTA[0]= temp_delta;
4031                               OILREC_ALT [0] =  temp_oil_rec_ALT;
4032                               RECEMUL_ALT[0] =  temp_smth_rec_ALT;
4033                              }
4034
4035                                  if (stage == 11 && pumprun_3 == 31) //alternate flooding [emulsion]
4036                                  {
4037                                   TEMPSTEP[0] = temp_time;
4038                                   TEMPDELTA[0]= temp_delta;
4039                                   OILREC_ALT [0] =  temp_oil_rec_ALT;
4040                                   RECEMUL_ALT[0] =  temp_smth_rec_ALT;
4041                                  }
4042
4043 CmtReleaseLock (Mass_lockHandle);
4044
4045 ProcessSystemEvents ();
4046 }
4047
4048 void DSN_SHIFT_MASS(int instrument)     //shifting data locally
4049 {
4050    switch (instrument)
4051    {
4052    case 0:
4053         Shift (M11, NUM, 1, M11);
4054         Shift (M12, NUM, 1, M12);
4055         Shift (M13, NUM, 1, M13);
4056         Shift (M14, NUM, 1, M14);
4057     break;
4058
4059    case 1:
4060         Shift (M21, NUM, 1, M21);
4061         Shift (M22, NUM, 1, M22);
4062         Shift (M23, NUM, 1, M23);
4063         Shift (M24, NUM, 1, M24);
4064     break;
4065
4066    case 2:
```

```
4067         Shift (M31, NUM, 1, M31);
4068         Shift (M32, NUM, 1, M32);
4069         Shift (M33, NUM, 1, M33);
4070         Shift (M34, NUM, 1, M34);
4071         Shift (Gtime_E, NUM, 1, Gtime_E);
4072 
4073                 if (stage >= 10 && pumprun_3 == 31 )        //emulsion injection into the cell
4074                 {
4075                     Shift (EMULIN, NUM, 1, EMULIN);
4076                     Shift (TEMPSTEP_E, NUM, 1, TEMPSTEP_E);
4077                     Shift (TEMPDELTA_E, NUM, 1, TEMPDELTA_E);
4078                 }
4079     break;
4080 
4081    case 3:
4082         Shift (M41, NUM, 1, M41);
4083         Shift (M42, NUM, 1, M42);
4084         Shift (M43, NUM, 1, M43);
4085         Shift (M44, NUM, 1, M44);
4086         Shift (Gtime, NUM, 1, Gtime);
4087 
4088                 if (stage == 4 && pumprun_2 == 21 )  // oil saturation process
4089                 {
4090                     Shift (WATERREC_ST9, NUM, 1, WATERREC_ST9);
4091                     Shift (OILREC_ST9, NUM, 1, OILREC_ST9);
4092                     Shift (TEMPSTEP, NUM, 1, TEMPSTEP);
4093                     Shift (TEMPDELTA, NUM, 1, TEMPDELTA);
4094 
4095                     Shift (SWI, NUM, 1, SWI);
4096                 }
4097 
4098                 if (stage == 7 && pumprun_1 == 11 )  // native waterflooding
4099                 {
4100                     Shift (WATERREC, NUM, 1, WATERREC);
4101                     Shift (OILREC, NUM, 1, OILREC);
4102                     Shift (TEMPSTEP, NUM, 1, TEMPSTEP);
4103                     Shift (TEMPDELTA, NUM, 1, TEMPDELTA);
4104                     Shift (WATERFLOODINJ, NUM, 1, WATERFLOODINJ);
4105 
4106                     Shift (SWI, NUM, 1, SWI);
4107                 }
4108 
4109                 if (stage == 10 && pumprun_3 == 31 ) // emulsion flooding
4110                 {
4111                     Shift (RECEMUL, NUM, 1, RECEMUL);
4112                     Shift (OILREC_ST14, NUM, 1, OILREC_ST14);
4113                     Shift (TEMPSTEP, NUM, 1, TEMPSTEP);
4114                     Shift (TEMPDELTA, NUM, 1, TEMPDELTA);
4115                 }
4116 
4117                 if (stage == 11 && pumprun_1 == 11 ) // alternate flooding [water]
4118                 {
4119                     Shift (RECEMUL_ALT, NUM, 1, RECEMUL_ALT);
4120                     Shift (OILREC_ALT, NUM, 1, OILREC_ALT);
4121                     Shift (TEMPSTEP, NUM, 1, TEMPSTEP);
4122                     Shift (TEMPDELTA, NUM, 1, TEMPDELTA);
4123                 }
4124 
4125                 if (stage == 11 && pumprun_3 == 31 ) // alternate flooding [emulsion]
4126                 {
4127                     Shift (RECEMUL_ALT, NUM, 1, RECEMUL_ALT);
4128                     Shift (OILREC_ALT, NUM, 1, OILREC_ALT);
4129                     Shift (TEMPSTEP, NUM, 1, TEMPSTEP);
4130                     Shift (TEMPDELTA, NUM, 1, TEMPDELTA);
4131                 }
4132     break;
4133    }
4134 return;
4135 }
4136 
4137 int CVICALLBACK BPR_ThreadFunction (void *functionData)
4138 {
4139 int conreq_BPR;
4140 
4141 while (BPR_quitflag == 1)
4142 {
4143       ProcessSystemEvents ();
4144 
4145       GetCtrlVal(config_handle,CONFIG_CON_BPR,&conreq_BPR);
4146 
4147             if(conreq_BPR == 1)
4148             {
4149                    ConnectToDDEServer (&hConv5, "FlowDDE5", "C(1)", DDECallback, 0);
4150                    conreq_BPR=0; Is_BPR=1;
4151                    SetCtrlVal(config_handle,CONFIG_CON_BPR, conreq_BPR);
4152                    SetCtrlVal(panel_handle,MAIN_LOG_BPR, Is_BPR);
4153                    SetCtrlVal (config_handle, CONFIG_BPR_LED, 1);
4154                    Update_Graphics();
4155             }
4156             else if (conreq_BPR == -1)
4157             {
4158                    DisconnectFromDDEServer (hConv5);
4159                    conreq_BPR=0; Is_BPR=0;
4160                    SetCtrlVal(config_handle,CONFIG_CON_BPR, conreq_BPR);
4161                    SetCtrlVal(panel_handle,MAIN_LOG_BPR, Is_BPR);
4162                    SetCtrlVal (config_handle, CONFIG_BPR_LED, 0);
4163                    Update_Graphics();
```

```
            }
                if (Is_BPR) READ_BPR();

      ProcessSystemEvents ();
}
return 0;
}

void READ_BPR(void)
{
char   temp[NUM];
double data,
       data2,
       data3;

     //fsetpoint [BPR1 pressure set point]
     GetCtrlVal(panel_handle,MAIN_BPR_SET,&bpr_set);
     if (bpr_set_was != bpr_set )
     {
         if (bpr_set >= 1449)
         {
             MessagePopup("Error","Pressure limited by 1450 psi");
             SetCtrlVal(panel_handle,MAIN_BPR_SET,1440.0);
             GetCtrlVal(panel_handle,MAIN_BPR_SET,&bpr_set);

             snprintf(temp, NUM, "%g", bpr_set);
             ClientDDEWrite(hConv5, "P(206)", CF_TEXT, temp, 64, 0);
             SetCtrlVal (panel_handle, MAIN_LED_BPR_CHECK, 1);
             Delay (0.5);
             SetCtrlVal (panel_handle, MAIN_LED_BPR_CHECK, 0);
             bpr_set_was = 1440;
         }
         else
         {
         snprintf(temp, NUM, "%g", bpr_set);
         ClientDDEWrite(hConv5, "P(206)", CF_TEXT, temp, 64, 0);
         SetCtrlVal (panel_handle, MAIN_LED_BPR_CHECK, 1);
         Delay (0.5);
         SetCtrlVal (panel_handle, MAIN_LED_BPR_CHECK, 0);
         bpr_set_was = bpr_set;
         }
     }

ProcessSystemEvents ();

    //fmeasure [BPR1 pressure measure point]
    Delay(0.1);
    ClientDDERead (hConv5, "P(205)", CF_TEXT, read_bpr11data, 64, 0);
    data=atof(read_bpr11data);
    BPR1TEMP[0] = data;
    SetCtrlVal (panel_handle, MAIN_BPR1, data);

    //fsetpoint read
    Delay(0.1);
    ClientDDERead (hConv5, "P(206)", CF_TEXT, read_bpr12data, 64, 0);
    data2=atof(read_bpr12data);
    SetCtrlVal (panel_handle, MAIN_BPR_SETPOINT, data2);

    //alarm info
    Delay(0.1);
    ClientDDERead (hConv5, "P(28)", CF_TEXT, read_bpr13data, 64, 0);
    data3=atof(read_bpr13data);
    SetCtrlVal (panel_handle, MAIN_BPR_ALARM, data3);

    CmtGetLock (BPR_lockHandle);

       Shift (BPR11, NUM, 1, BPR11);
       Shift (BPR12, NUM, 1, BPR12);
       Shift (BPR13, NUM, 1, BPR13);

       BPR11[0] = data;
       BPR12[0] = data2;
       BPR13[0] = data3;

    CmtReleaseLock (BPR_lockHandle);

ProcessSystemEvents ();
}

//-------------------------------------------------------------
// DSN_Setup_DAQ    : Sets up the DAQ Task and RTD Channels
//-------------------------------------------------------------

void DSN_Setup_DAQ(void)
{
//  int i,count=0;

        // Retrieve Averaging Number and Sampling Rate from Panel

        GetCtrlVal(config_handle,CONFIG_DAQ_AVERAGING_NUMBER,&DAQsampsPerChan);
        GetCtrlVal(config_handle,CONFIG_DAQ_RATE,&DAQrate);

        GetCtrlVal(config_handle,CONFIG_TP2CHANNEL,TP2channel);

//      GetCtrlVal(config_handle,CONFIG_DP1CHANNEL_A,DP1channel_A);
//      GetCtrlVal(config_handle,CONFIG_DP1CHANNEL_B,DP1channel_B);

```

```
4261         GetCtrlVal(config_handle,CONFIG_TP1CHANNEL,TP1channel);
4262
4263
4264         GetCtrlVal(config_handle,CONFIG_CP11_A,CP11_Achannel);
4265         GetCtrlVal(config_handle,CONFIG_CP11_B,CP11_Bchannel);
4266         GetCtrlVal(config_handle,CONFIG_CP12_A,CP12_Achannel);
4267         GetCtrlVal(config_handle,CONFIG_CP12_B,CP12_Bchannel);
4268         GetCtrlVal(config_handle,CONFIG_CP13_A,CP13_Achannel);
4269         GetCtrlVal(config_handle,CONFIG_CP13_B,CP13_Bchannel);
4270         GetCtrlVal(config_handle,CONFIG_CP14_A,CP14_Achannel);
4271         GetCtrlVal(config_handle,CONFIG_CP14_B,CP14_Bchannel);
4272         GetCtrlVal(config_handle,CONFIG_CP15_A,CP15_Achannel);
4273         GetCtrlVal(config_handle,CONFIG_CP15_B,CP15_Bchannel);
4274
4275         GetCtrlVal(config_handle,CONFIG_CP21_A,CP21_Achannel);
4276         GetCtrlVal(config_handle,CONFIG_CP21_B,CP21_Bchannel);
4277         GetCtrlVal(config_handle,CONFIG_CP22_A,CP22_Achannel);
4278         GetCtrlVal(config_handle,CONFIG_CP22_B,CP22_Bchannel);
4279         GetCtrlVal(config_handle,CONFIG_CP23_A,CP23_Achannel);
4280         GetCtrlVal(config_handle,CONFIG_CP23_B,CP23_Bchannel);
4281         GetCtrlVal(config_handle,CONFIG_CP24_A,CP24_Achannel);
4282         GetCtrlVal(config_handle,CONFIG_CP24_B,CP24_Bchannel);
4283         GetCtrlVal(config_handle,CONFIG_CP25_A,CP25_Achannel);
4284         GetCtrlVal(config_handle,CONFIG_CP25_B,CP25_Bchannel);
4285
4286         GetCtrlVal(config_handle,CONFIG_CP31_A,CP31_Achannel);
4287         GetCtrlVal(config_handle,CONFIG_CP31_B,CP31_Bchannel);
4288         GetCtrlVal(config_handle,CONFIG_CP32_A,CP32_Achannel);
4289         GetCtrlVal(config_handle,CONFIG_CP32_B,CP32_Bchannel);
4290         GetCtrlVal(config_handle,CONFIG_CP33_A,CP33_Achannel);
4291         GetCtrlVal(config_handle,CONFIG_CP33_B,CP33_Bchannel);
4292         GetCtrlVal(config_handle,CONFIG_CP34_A,CP34_Achannel);
4293         GetCtrlVal(config_handle,CONFIG_CP34_B,CP34_Bchannel);
4294         GetCtrlVal(config_handle,CONFIG_CP35_A,CP35_Achannel);
4295         GetCtrlVal(config_handle,CONFIG_CP35_B,CP35_Bchannel);
4296
4297
4298         //*****************************************/
4299         // Create the DAQ task       /
4300         //*****************************************/
4301
4302         DAQmxErrChk (DAQmxCreateTask("",&taskHandle));
4303
4304         //*****************************************/
4305         // Create all the Necessary Channels         /
4306         //*****************************************/
4307
4308                 // Sensor is a Pressure Transducer TP-2
4309                                                         //  "cDAQ1Mod5/ai11"
4310                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, TP2channel ,
4311                                 "Voltage_1A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4312
4313                 // Sensor is a Pressure Transducer dP1
4314
4315                                                         //  "cDAQ1Mod5/ai20"
4316 //              DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, DP1channel_A,
4317 //                              "Voltage_1B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4318
4319                                                         //  "cDAQ1Mod5/ai21"
4320 //              DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, DP1channel_B,
4321 //                              "Voltage_1C", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4322
4323                 // Sensor is a Pressure Transducer TP-1
4324                                                         //  "cDAQ1Mod5/ai16"
4325                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, TP1channel,
4326                                 "Voltage_1D", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4327
4328 //Pressure sensors 1.1 ,1.2, 1.3, 1.4, 1.5
4329                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP11_Achannel,
4330                                 "Voltage_11_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4331                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP11_Bchannel,
4332                                 "Voltage_11_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4333                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP12_Achannel,
4334                                 "Voltage_12_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4335                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP12_Bchannel,
4336                                 "Voltage_12_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4337                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP13_Achannel,
4338                                 "Voltage_13_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4339                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP13_Bchannel,
4340                                 "Voltage_13_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4341                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP14_Achannel,
4342                                 "Voltage_14_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4343                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP14_Bchannel,
4344                                 "Voltage_14_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4345                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP15_Achannel,
4346                                 "Voltage_15_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4347                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP15_Bchannel,
4348                                 "Voltage_15_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4349
4350 //Pressure sensors 2.1 ,2.2, 2.3, 2.4, 2.5
4351                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP21_Achannel,
4352                                 "Voltage_21_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4353                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP21_Bchannel,
4354                                 "Voltage_21_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4355                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP22_Achannel,
4356                                 "Voltage_22_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
4357                 DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP22 Bchannel,
```

```
                            "Voltage_22_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP23_Achannel,
                            "Voltage_23_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP23_Bchannel,
                            "Voltage_23_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP24_Achannel,
                            "Voltage_24_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP24_Bchannel,
                            "Voltage_24_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP25_Achannel,
                            "Voltage_25_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP25_Bchannel,
                            "Voltage_25_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));

//Pressure sensors 3.1 ,3.2, 3.3, 3.4, 3.5
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP31_Achannel,
                            "Voltage_31_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP31_Bchannel,
                            "Voltage_31_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP32_Achannel,
                            "Voltage_32_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP32_Bchannel,
                            "Voltage_32_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP33_Achannel,
                            "Voltage_33_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP33_Bchannel,
                            "Voltage_33_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP34_Achannel,
                            "Voltage_34_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP34_Bchannel,
                            "Voltage_34_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP35_Achannel,
                            "Voltage_35_A", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));
                DAQmxErrChk(DAQmxCreateAIVoltageChan(taskHandle, CP35_Bchannel,
                            "Voltage_35_B", DAQmx_Val_RSE, 0, 10, DAQmx_Val_Volts, ""));

    //*********************************************/
    // Setup the DAQ Timing and Number of Channels  /
    //*********************************************/

    // Set the clock timing
    DAQmxErrChk (DAQmxCfgSampClkTiming (taskHandle,"", DAQrate,
        DAQmx_Val_Rising, DAQmx_Val_ContSamps, DAQsampsPerChan));

    // Set the number of channels to scan
    DAQmxErrChk (DAQmxGetTaskAttribute(taskHandle,DAQmx_Task_NumChans,&DAQnumChannels));

    // Make space for the data
    if( (DAQdata=malloc(DAQsampsPerChan*DAQnumChannels*sizeof(float64)))==NULL )
        {
        MessagePopup("Error","Not enough memory");
        goto Error;
        }
    // Make space for the averaging aray
    if( (DAQArray_Out=malloc(DAQsampsPerChan*sizeof(float64)))==NULL )
        {
        MessagePopup("Error","Not enough memory");
        goto Error;
        }

    //Exit
    Error:
        if( DAQmxFailed(DAQerror) )
            {
                DAQmxGetExtendedErrorInfo(DAQerrBuff,2048);
                DAQquitflag =0;
            }

        if( DAQmxFailed(DAQerror) )
                MessagePopup("DAQmx Error",DAQerrBuff);
}

/*************************************************************/
/* DAQThreadFunction ():    - Separate Thread for DAQ          */
/*          - Logs to file/graphs*/
/*************************************************************/
int CVICALLBACK DAQThreadFunction (void *functionData)
{  // double T1; // T2,Hz;

//  int i, j, k = 0;
//  int count = 0;

//  Start a loop that will process events for this thread
    while (DAQquitflag == 1)
    {
            ++SAMPLES;


        //  T1 = Timer ();// Start TIME

            ProcessSystemEvents ();

            //-----------------------------------------------


            step[0] = Timer ();         // Get Time

```

```
            if(Is_DAQ)
            {
                DSN_Run_DAQ();
            }

            //---------------------------------------------

            ProcessSystemEvents ();
            SetCtrlVal(panel_handle, MAIN_SAMPLES, SAMPLES);

            //---------------------------------------------

            LiquidsInj();

            Perm();

            //TIME STAMP
            GetCurrentCVIAbsoluteTime (&timestamp);

            CVIAbsoluteTimeToCVIUILTime (timestamp, &timeDAQ);

            if(Is_Log)
                LogFile();          // Log to File

            ProcessSystemEvents ();

            CmtGetLock (DAQ_lockHandle);

            if(Is_Graph)
                Graph();            // Do Graphing

            CmtReleaseLock (DAQ_lockHandle);

            DSN_SHIFT_DAQ();            // SHIFT all DAQ arrays
            //---------------------------------------------

    }
return 0;
}

void LiquidsInj(void)
{
double temp, temp2, temp3, outlettemp, curttube;
int a;

//WATER
if (pumprun_1 == 11)
{
    temp=PFR11[0]*(step[0]-step[1])/60;
    G_WATERINJ[0] = G_WATERINJ[1] - temp;
    OUTLET[0]=OUTLET[1]+temp;

    if (stage >=0 && stage <=3) WATERINJ[0]=WATERINJ[1] + temp;  //PV

          if (stage == 3 )  WATERINJPV[0]=WATERINJ[0]/PV[0];    //saturation process

              if (stage >=7)     //waterflooding
              {
                  WATERINJ2[0]=WATERINJ2[1] + temp;
                  WATERINJ2PV[0]=WATERINJ2[0]/PV[0];
                  WATERINJ[0]=WATERINJ[1];
              }
SetCtrlVal (panel_handle, MAIN_WATERTANK, G_WATERINJ[0]);
SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK


GetCtrlAttribute (panel_handle, MAIN_OUTLETTANK,ATTR_MAX_VALUE, &outlettemp);
GetCtrlVal (panel_handle, MAIN_CUR_TTUBE, &curttube);

if (OUTLET[0] >= outlettemp && stage >= 7)
{//change to the new tube at fraction collector
        SWITCH_FRACTION (0, 1, 0, 0);
        Delay (0.5);
        SWITCH_FRACTION (0, 0, 0, 0);
    MANUAL_TEST_TUBE[0] = curttube + 1;
    SetCtrlVal(panel_handle, MAIN_CUR_TTUBE, MANUAL_TEST_TUBE[0]);

    globalemail = 1;   //set command to send email


    // clean the outlet
    OUTLET[1]=0.0;
    OUTLET[0]=0.0;
    SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
}
return;
}
if (pumprun_1 == 12)
{

//temp for time test
//temp=PFR11[0]*(step[0]-step[1])/60;
//WATERINJ[0]=WATERINJ[1] + temp;
//SetCtrlVal (panel_handle, MAIN_WATER_TEST, WATERINJ[0]);

            G_WATERINJ[0]=G_WATERINJ[1];
            WATERINJ2[0]=WATERINJ2[1];
```

```
4552             WATERINJ[0]=WATERINJ[1];
4553             OUTLET[0]=OUTLET[1];
4554
4555          if (stage == 3)  WATERINJPV[0]=WATERINJPV[1];
4556          if (stage >= 7)  WATERINJ2PV[0]=WATERINJ2PV[1];
4557
4558 SetCtrlVal (panel_handle, MAIN_WATERTANK,  G_WATERINJ[0]);
4559 SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4560 }
4561
4562 //OIL
4563 if (pumprun_2 == 21)
4564 {
4565             temp2=PFR21[0]*(step[0]-step[1])/60;
4566             OILINJ[0]=OILINJ[1] + temp2;
4567             G_OILINJ[0] = G_OILINJ[1] - temp2;
4568             OUTLET[0]=OUTLET[1]+temp2;
4569
4570
4571             if (stage >= 3)  OILINJPV[0]=OILINJ[0]/PV[0];
4572
4573 SetCtrlVal (panel_handle, MAIN_OILTANK,  G_OILINJ[0]);
4574 SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4575
4576   return;
4577 }
4578
4579 if (pumprun_2 == 22)
4580 {
4581                  G_OILINJ[0]= G_OILINJ[1];
4582                  OILINJ[0]=OILINJ[1];
4583                  OUTLET[0]=OUTLET[1];
4584
4585                  if (stage >= 3)  OILINJPV[0]=OILINJPV[1];
4586
4587                      SetCtrlVal (panel_handle, MAIN_OILTANK,  G_OILINJ[0]);
4588                      SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4589
4590 }
4591
4592
4593                    //EMULSION
4594                    if (pumprun_3 == 31)
4595                    {
4596
4597                        temp3=PFR31[0]*(step[0]-step[1])/60;
4598                        EMULINJ[0]=EMULINJ[1] + temp3;
4599                        G_EMULINJ[0]= G_EMULINJ[1] - temp3 ;
4600                        OUTLET[0]=OUTLET[1]+temp3;
4601
4602                        if (stage >= 3)  EMULINJPV[0]=EMULINJ[0]/PV[0];
4603
4604                        SetCtrlVal (panel_handle, MAIN_EMULTANK, G_EMULINJ[0]);
4605                        SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4606
4607 GetCtrlAttribute (panel_handle, MAIN_OUTLETTANK,ATTR_MAX_VALUE, &outlettemp);
4608 GetCtrlVal (panel_handle, MAIN_CUR_TTUBE, &curttube);
4609
4610 if (OUTLET[0] >= outlettemp && stage >= 7)
4611 {//change to the new tube at fraction collector
4612         SWITCH_FRACTION (0, 1, 0, 0);
4613         Delay (0.5);
4614         SWITCH_FRACTION (0, 0, 0, 0);
4615     MANUAL_TEST_TUBE[0] = curttube + 1;
4616     SetCtrlVal(panel_handle, MAIN_CUR_TTUBE, MANUAL_TEST_TUBE[0]);
4617     // clean the outlet
4618     OUTLET[1]=0.0;
4619     OUTLET[0]=0.0;
4620     SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4621 }
4622                     return;
4623                     }
4624                         if (pumprun_3 == 32)
4625                         {
4626                              EMULINJ[0]=EMULINJ[1];
4627                              G_EMULINJ[0]=G_EMULINJ[1];
4628                              OUTLET[0]=OUTLET[1];
4629
4630                              if (stage >= 3)  EMULINJPV[0]=EMULINJPV[1];
4631
4632                              SetCtrlVal (panel_handle, MAIN_EMULTANK, G_EMULINJ[0]);
4633                              SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4634
4635                         }
4636
4637 SetCtrlVal (panel_handle, MAIN_WATERTANK, G_WATERINJ[0]);
4638 SetCtrlVal (panel_handle, MAIN_OILTANK,   G_OILINJ[0]);
4639 SetCtrlVal (panel_handle, MAIN_EMULTANK,  G_EMULINJ[0]);
4640 SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK
4641
4642
4643
4644 return;
4645 }
4646
4647 void Perm(void)
4648 {
```

```
4649 double data,data2,data3;
4650 double showdata_11,showdata_12,showdata_21,showdata_22,showdata_31,showdata_32;
4651
4652 //Water
4653 if (pumprun_1 == 11)
4654 {
4655 data = Perm_coef * PFR11[0] * VIS_WATER;
4656
4657 PERMWATERDP1[0] = data / DP1[0];  //Darcy
4658
4659 PERMWATERDELTA[0] = data / Delta[0];  //Darcy
4660
4661 showdata_11 = permcoef * PERMWATERDP1[0];
4662 showdata_12 = permcoef * PERMWATERDELTA[0];
4663
4664 SetCtrlVal(panel_handle, MAIN_PERM_W_1,showdata_11);
4665 SetCtrlVal(panel_handle, MAIN_PERM_W_2,showdata_12);
4666
4667 return;
4668 }
4669     if (pumprun_1 == 12)
4670     {
4671         PERMWATERDP1[0]=0;
4672         PERMWATERDELTA[0]=0;
4673         showdata_11 = permcoef * PERMWATERDP1[0];
4674         showdata_12 = permcoef * PERMWATERDELTA[0];
4675
4676         SetCtrlVal(panel_handle, MAIN_PERM_W_1,showdata_11);
4677         SetCtrlVal(panel_handle, MAIN_PERM_W_2,showdata_12);
4678
4679
4680     }
4681
4682             //Oil
4683             if (pumprun_2 == 21)
4684             {
4685             data2= Perm_coef * PFR21[0] *  VIS_OIL;
4686
4687             PERMOILDP1[0] = data2 / DP1[0]; //Darcy
4688
4689             PERMOILDELTA[0] = data2 / Delta[0]; //Darcy
4690
4691             showdata_21 = permcoef * PERMOILDP1[0];
4692
4693             showdata_22 = permcoef * PERMOILDELTA[0];
4694
4695             SetCtrlVal(panel_handle, MAIN_PERM_O_1,showdata_21);
4696             SetCtrlVal(panel_handle, MAIN_PERM_O_2,showdata_22);
4697             return;
4698             }
4699
4700                 if (pumprun_2 == 22)
4701                 {
4702                 PERMOILDP1[0]=0;
4703                 PERMOILDELTA[0]=0;
4704
4705                 showdata_21 = permcoef * PERMOILDP1[0];
4706                 showdata_22 = permcoef * PERMOILDELTA[0];
4707
4708                 SetCtrlVal(panel_handle, MAIN_PERM_O_1,showdata_21);
4709                 SetCtrlVal(panel_handle, MAIN_PERM_O_2,showdata_22);
4710
4711                 }
4712
4713                         //Emulsion
4714                         if (pumprun_3 == 31)
4715                         {
4716                         data3 = Perm_coef * PFR31[0] * VIS_EMUL;
4717
4718                         PERMEMULDP1[0] = data3 / DP1[0];
4719                         PERMEMULDELTA[0] = data3 / Delta[0];
4720
4721                         showdata_31 = permcoef * PERMEMULDP1[0];
4722                         showdata_32 = permcoef * PERMEMULDELTA[0];
4723
4724                         SetCtrlVal(panel_handle, MAIN_PERM_E_1,showdata_31);
4725                         SetCtrlVal(panel_handle, MAIN_PERM_E_2,showdata_32);
4726                         return;
4727                         }
4728
4729
4730                             if (pumprun_3 == 32)
4731                             {
4732                             PERMEMULDP1[0]=0;
4733                             PERMEMULDELTA[0]=0;
4734
4735                             showdata_31 = permcoef * PERMEMULDP1[0];
4736                             showdata_32 = permcoef * PERMEMULDELTA[0];
4737
4738                             SetCtrlVal(panel_handle, MAIN_PERM_E_1,showdata_31);
4739                             SetCtrlVal(panel_handle, MAIN_PERM_E_2,showdata_32);
4740
4741                             }
4742
4743 return;
4744 }
4745
```

```
4746 //-------------------------------------------------------------
4747 // DSN_Run_DAQ  : Run the DAQ Task and RTD Channels
4748 //-------------------------------------------------------------
4749
4750 void DSN_Run_DAQ(void)     // Gettig data from DAQ for pressure sensors
4751 {
4752     int i;
4753     double pressure_scaling_factor_P  = 150;    // psi per Volt
4754     double pressure_scaling_factor_dP = 15;      // psi per Volt
4755     double pressure_scaling_factor_dP_120 = 12;      // psi per Volt
4756     double pressure_scaling_factor_dP_80 = 8;      // psi per Volt
4757
4758
4759
4760 //  int    error=0;
4761
4762     //--------------------------------------------------------------
4763     // DAQmx Start DAQ Task       /
4764     //--------------------------------------------------------------
4765
4766     if(DAQ_Task_Started)
4767     {
4768         // Do not attempt to start task
4769     }
4770     else
4771     {
4772         DAQmxErrChk (DAQmxStartTask(taskHandle));
4773     }
4774
4775     DAQ_Task_Started = 1;
4776     ProcessDrawEvents();
4777
4778     //--------------------------------------------------------------
4779     // DAQmx Read Code            /
4780     //--------------------------------------------------------------
4781
4782     Clear1D (DAQdata, DAQsampsPerChan*DAQnumChannels);
4783
4784     DAQmxReadAnalogF64(taskHandle,DAQsampsPerChan,10.0,DAQmx_Val_GroupByChannel,
4785         DAQdata,DAQsampsPerChan*DAQnumChannels,&DAQnumRead,NULL);
4786
4787 if( DAQnumRead>0 )
4788 for(i=0;i<DAQnumChannels;i++)
4789 {
4790 // Extract Data for one Channel
4791 Subset1D (DAQdata, DAQnumChannels*DAQnumRead, i*DAQnumRead, DAQnumRead, DAQArray_Out);
4792
4793 // Average the Data Points
4794 Mean (DAQArray_Out,DAQnumRead, &DAQoutMean);
4795
4796     switch(i)
4797     {
4798             case 0:   //upstream
4799                 DAQoutMean = DAQoutMean * pressure_scaling_factor_P - ZEROTP2[0];
4800                 SetCtrlVal(panel_handle,MAIN_TP2,DAQoutMean);
4801                 TP2[0] = DAQoutMean;
4802                     //Danger for membrane
4803                      if (TP2[0] >= 1450.00)
4804                      {
4805                      STOPPUMPS();  //will stop pumps which are in use
4806                      SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_2,ATTR_VISIBLE, 1);
4807                      SetCtrlAttribute(panel_handle,MAIN_TP2_OVER_1450,ATTR_VISIBLE, 1);
4808                      }
4809                 break;
4810 //          case 1:
4811 //              DAQoutMean = DAQoutMean * pressure_scaling_factor_dP;
4812 //              DP1_A[0] = DAQoutMean;
4813 //              break;
4814 //          case 2:
4815 //              DAQoutMean = DAQoutMean * pressure_scaling_factor_dP;
4816 //              DP1_B[0] = DAQoutMean;
4817 //              break;
4818             case 1:  //downstream
4819                 DAQoutMean = DAQoutMean * pressure_scaling_factor_P - ZEROTP1[0];
4820                 SetCtrlVal(panel_handle,MAIN_TP1,DAQoutMean);
4821                 TP1[0] = DAQoutMean;
4822                     //Danger for membrane
4823                      if (TP1[0] >= 1450.00)
4824                      {
4825                      STOPPUMPS();  //will stop pumps which are in use
4826                      SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_7,ATTR_VISIBLE, 1);
4827                      SetCtrlAttribute(panel_handle,MAIN_TP1_OVER_1450,ATTR_VISIBLE, 1);
4828                      }
4829                 break;
4830             case 2:
4831                 DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4832                 CP11_A[0] = DAQoutMean;
4833             break;
4834             case 3:
4835                 DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4836                 CP11_B[0] = DAQoutMean;
4837                 break;
4838                 case 4:
4839                     DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4840                     CP12_A[0] = DAQoutMean;
4841                 break;
4842                 case 5:
```

```
4843                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4844                    CP12_B[0] = DAQoutMean;
4845                    break;
4846                    case 6:
4847                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4848                            CP13_A[0] = DAQoutMean;
4849                            break;
4850                        case 7:
4851                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4852                            CP13_B[0] = DAQoutMean;
4853                            break;
4854                                case 8:
4855                                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4856                                    CP14_A[0] = DAQoutMean;
4857                                    break;
4858                                case 9:
4859                                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4860                                    CP14_B[0] = DAQoutMean;
4861                                    break;
4862                                        case 10:
4863                                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4864                                        CP15_A[0] = DAQoutMean;
4865                                        break;
4866                                        case 11:
4867                                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4868                                            CP15_B[0] = DAQoutMean;
4869                                            break;
4870            case 12:
4871                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4872                CP21_A[0] = DAQoutMean;
4873                break;
4874            case 13:
4875                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4876                CP21_B[0] = DAQoutMean;
4877                break;
4878                case 14:
4879                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4880                    CP22_A[0] = DAQoutMean;
4881                    break;
4882                    case 15:
4883                        DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4884                        CP22_B[0] = DAQoutMean;
4885                        break;
4886                        case 16:
4887                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4888                            CP23_A[0] = DAQoutMean;
4889                            break;
4890                            case 17:
4891                                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4892                                CP23_B[0] = DAQoutMean;
4893                                break;
4894                                case 18:
4895                                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4896                                    CP24_A[0] = DAQoutMean;
4897                                    break;
4898                                    case 19:
4899                                        DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4900                                        CP24_B[0] = DAQoutMean;
4901                                        break;
4902                                        case 20:
4903                                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4904                                            CP25_A[0] = DAQoutMean;
4905                                            break;
4906                                        case 21:
4907                                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4908                                            CP25_B[0] = DAQoutMean;
4909                                            break;
4910            case 22:
4911                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4912                CP31_A[0] = DAQoutMean;
4913                break;
4914            case 23:
4915                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4916                CP31_B[0] = DAQoutMean;
4917                break;
4918                case 24:
4919                    DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4920                    CP32_A[0] = DAQoutMean;
4921                    break;
4922                    case 25:
4923                        DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_120;
4924                        CP32_B[0] = DAQoutMean;
4925                        break;
4926                        case 26:
4927                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4928                            CP33_A[0] = DAQoutMean;
4929                        break;
4930                        case 27:
4931                            DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4932                            CP33_B[0] = DAQoutMean;
4933                            break;
4934                            case 28:
4935                                DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
4936                                CP34_B[0] = DAQoutMean;
4937                                break;
4938                                case 29:
4939                                    DAQoutMean = DAQoutMean * pressure scaling factor dP 80;
```

```
                                    CP34_B[0] = DAQoutMean;
                                    break;
                                    case 30:
                                        DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
                                        CP35_A[0] = DAQoutMean;
                                        break;
                                    case 31:
                                        DAQoutMean = DAQoutMean * pressure_scaling_factor_dP_80;
                                        CP35_B[0] = DAQoutMean;
                                        break;
}

//  DP1[0]=DP1_B[0]-DP1_A[0] - ZERODP1[0];
//  SetCtrlVal(panel_handle,MAIN_DP1,DP1[0]);

    CP11[0]=CP11_B[0]-CP11_A[0] - ZEROCP11[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP1_1,CP11[0]);
    CP12[0]=CP12_B[0]-CP12_A[0] - ZEROCP12[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP1_2,CP12[0]);
    CP13[0]=CP13_B[0]-CP13_A[0] - ZEROCP13[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP1_3,CP13[0]);
    CP14[0]=CP14_B[0]-CP14_A[0] - ZEROCP14[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP1_4,CP14[0]);
    CP15[0]=CP15_B[0]-CP15_A[0] - ZEROCP15[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP1_5,CP15[0]);

    CP21[0]=CP21_B[0]-CP21_A[0] - ZEROCP21[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP2_1,CP21[0]);
    CP22[0]=CP22_B[0]-CP22_A[0] - ZEROCP22[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP2_2,CP22[0]);
    CP23[0]=CP23_B[0]-CP23_A[0] - ZEROCP23[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP2_3,CP23[0]);
    CP24[0]=CP24_B[0]-CP24_A[0] - ZEROCP24[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP2_4,CP24[0]);
    CP25[0]=CP25_B[0]-CP25_A[0] - ZEROCP25[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP2_5,CP25[0]);

    CP31[0]=CP31_B[0]-CP31_A[0] - ZEROCP31[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP3_1,CP31[0]);
    CP32[0]=CP32_B[0]-CP32_A[0] - ZEROCP32[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP3_2,CP32[0]);
    CP33[0]=CP33_B[0]-CP33_A[0] - ZEROCP33[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP3_3,CP33[0]);
    CP34[0]=CP34_B[0]-CP34_A[0] - ZEROCP34[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP3_4,CP34[0]);
    CP35[0]=CP35_B[0]-CP35_A[0] - ZEROCP35[0];
    SetCtrlVal(pressure_handle,PRESSURE_CP3_5,CP35[0]);

    //Danger for membrane
            if (DP1[0] >= 140.00  && OVERPRES == 1)
            {
            SetCtrlVal(panel_handle, MAIN_AV2_ONOFF, 0);
            SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_1,ATTR_VISIBLE, 1);
            SetCtrlAttribute(panel_handle,MAIN_DIFF_OVER_150,ATTR_VISIBLE, 1);
            SWITCH_SOLENOID ();
            OVERPRES = 0;
            }

//Delta[0]=TP2[0]-BPR1TEMP[0]; removed from software, since new pressure sensor was added November 22,2013
Delta[0]=TP2[0]-TP1[0];

SetCtrlVal(panel_handle,MAIN_DP1_2,Delta[0]);
}
//Exit
Error:
    if( DAQmxFailed(DAQerror) )
        {
            DAQmxGetExtendedErrorInfo(DAQerrBuff,2048);
            DAQquitflag =0;
        }
}

//*****************************************************
//  DSN_SHIFT_DAQ   : Will step data along an array
//  : For all data that can be plotted
//*****************************************************
void DSN_SHIFT_DAQ(void)
{
Shift (step,    NUM, 1,step);

Shift (OILINJ,    NUM, 1,OILINJ);
Shift (OILINJPV,    NUM, 1,OILINJPV);

Shift (EMULINJ,   NUM, 1,EMULINJ);
Shift (EMULINJPV,   NUM, 1,EMULINJPV);

if (stage >=0 && stage <=3)
            Shift (WATERINJ,  NUM, 1,WATERINJ);

    if (stage == 3)
            Shift (WATERINJPV,  NUM, 1,WATERINJPV);  //saturation

            if (stage >= 7)
            {
                Shift (WATERINJ,  NUM, 1,WATERINJ);
                Shift (WATERINJ2,  NUM, 1,WATERINJ2);
                Shift (WATERINJ2PV,  NUM, 1,WATERINJ2PV);
```

```
5037            }
5038
5039     Shift (PERMWATERDP1,NUM, 1,PERMWATERDP1);
5040     Shift (PERMWATERDELTA,  NUM, 1,PERMWATERDELTA);
5041
5042     Shift (PERMOILDP1, NUM, 1,PERMOILDP1);
5043     Shift (PERMOILDELTA,NUM, 1,PERMOILDELTA);
5044
5045     Shift (PERMEMULDP1,  NUM, 1,PERMEMULDP1);
5046     Shift (PERMEMULDELTA, NUM, 1,PERMEMULDELTA);
5047
5048 if (G_WATERINJ[0] =! G_WATERINJ[1])
5049                                 Shift (G_WATERINJ,  NUM, 1,G_WATERINJ);
5050 if (G_OILINJ[0] =! G_OILINJ[1])
5051                                 Shift (G_OILINJ,  NUM, 1,G_OILINJ);
5052 if (G_EMULINJ[0] =! G_EMULINJ[1])
5053                                 Shift (G_EMULINJ,  NUM, 1,G_EMULINJ);
5054
5055     Shift (OUTLET,    NUM, 1,OUTLET);
5056     Shift (TP2,       NUM, 1,TP2);  //inlet
5057     Shift (DP1,       NUM, 1,DP1);  //dP-1 sensor
5058     Shift (TP1,       NUM, 1,TP1);  //outlet
5059     Shift (Delta,     NUM, 1,Delta); // TP2-TP1 delta pressure
5060
5061     Shift (CP11,       NUM, 1,CP11);  //inlet
5062     Shift (CP12,       NUM, 1,CP12);  //inlet
5063     Shift (CP13,       NUM, 1,CP13);  //inlet
5064     Shift (CP14,       NUM, 1,CP14);  //inlet
5065     Shift (CP15,       NUM, 1,CP15);  //inlet
5066
5067     Shift (CP21,       NUM, 1,CP21);  //inlet
5068     Shift (CP22,       NUM, 1,CP22);  //inlet
5069     Shift (CP23,       NUM, 1,CP23);  //inlet
5070     Shift (CP24,       NUM, 1,CP24);  //inlet
5071     Shift (CP25,       NUM, 1,CP25);  //inlet
5072
5073     Shift (CP31,       NUM, 1,CP31);  //inlet
5074     Shift (CP32,       NUM, 1,CP32);  //inlet
5075     Shift (CP33,       NUM, 1,CP33);  //inlet
5076     Shift (CP34,       NUM, 1,CP34);  //inlet
5077     Shift (CP35,       NUM, 1,CP35);  //inlet
5078
5079
5080     Shift (MANUAL_TEST_TUBE, NUM, 1, MANUAL_TEST_TUBE);
5081 return;
5082 }
5083
5084 //*******************************************************
5085 // DSN_SHIFT_Slow   : Will step data along an array
5086 //  : For all data that can be plotted
5087 //*******************************************************
5088
5089 void DSN_SHIFT_Slow(int instrument)
5090 {
5091     switch(instrument)
5092     {
5093         case 0:     // Pump #1 Volume flow rate
5094             Shift (PFR11,NUM, 1, PFR11);
5095             break;
5096         case 1:     // Pump #2 Volume flow rate
5097             Shift (PFR21,NUM, 1, PFR21);
5098             break;
5099         case 2:     // Pump #3 Volume flow rate
5100             Shift (PFR31,NUM, 1, PFR31);
5101             break;
5102         case 3:     // Pump #1 Pressure
5103             Shift (P11,NUM, 1, P11);
5104             break;
5105         case 4:     // Pump #2 Pressure
5106             Shift (P21,NUM, 1, P21);
5107             break;
5108         case 5:     // Pump #3 Pressure
5109             Shift (P31,NUM, 1, P31);
5110             break;
5111     }
5112 return;
5113 }
5114
5115 //*****************************************************************
5116 // DSN_Graph        : Calls Graphing Function and Scales X Axes
5117 // build based on DSN_Graph
5118 //*****************************************************************
5119 void Graph(void)
5120 {
5121 // GRAPH #1
5122     if(OnOff_G1)
5123     {
5124     Graph_Select(plotVar_G_1, MAIN_G_1);
5125         if(plotVar_G_1>=0)
5126         SetAxisScalingMode (panel_handle, MAIN_G_1, VAL_XAXIS, VAL_MANUAL, step[0]-X_Range_G_1, step[0]);
5127     }
5128 // GRAPH #2
5129     if(OnOff_G2)
5130     {
5131     Graph_Select(plotVar_G_2, MAIN_G_2);
5132         if(plotVar_G_2>=0)
5133         SetAxisScalingMode (panel_handle, MAIN G 2, VAL XAXIS, VAL MANUAL, step[0]-X Range G 2, step[0]);
```

```
5134     }
5135 // GRAPH #3
5136     if(OnOff_G3)
5137     {
5138     Graph_Select(plotVar_G_3, MAIN_G_3);
5139         if(plotVar_G_3>=0)
5140         SetAxisScalingMode (panel_handle, MAIN_G_3, VAL_XAXIS, VAL_MANUAL, step[0]-X_Range_G_3, step[0]);
5141     }
5142 // GRAPH #4
5143     if(OnOff_G4)
5144     {
5145     Graph_Select(plotVar_G_4, MAIN_G_4);
5146         if(plotVar_G_4>=0)
5147         SetAxisScalingMode (panel_handle, MAIN_G_4, VAL_XAXIS, VAL_AUTOSCALE, step[0]-X_Range_G_4, step[0]);
5148     }
5149 // GRAPH #5
5150     if(OnOff_G5)
5151     {
5152     Graph_Select(plotVar_G_5, EXTRA_G_G_5);
5153     }
5154
5155 // GRAPH #6 _ showing percentage of each liquid
5156     if(OnOff_G6)
5157     {
5158     plotVar_G_6=25;
5159     Graph_Select(plotVar_G_6, MAIN_G_6);
5160         if(plotVar_G_6>=0)
5161         SetAxisScalingMode (panel_handle, MAIN_G_6, VAL_XAXIS, VAL_AUTOSCALE, step[0]-250, step[0]);
5162     }
5163
5164 return;
5165 }
5166
5167 //*********************************************************
5168 // Graph_Select     : Plots the Selected Variable
5169 // build based on DSN_Graph_Select
5170 //*********************************************************
5171 void Graph_Select(int plotVar, int Panel_Graph)
5172 {
5173 switch(plotVar)
5174     {
5175         case -2:
5176             DeleteGraphPlot (extra_g_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5177             break;
5178         case -1:
5179             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5180             break;
5181         case 0: // Pressure [Pump #1 and TP-2]
5182             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5183
5184             PlotXY (panel_handle, Panel_Graph, step, P11, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5185             PlotXY (panel_handle, Panel_Graph, step, TP2, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,
     1, VAL_WHITE);
5186             break;
5187         case 1: // Pressure [Pump #3 and TP-2]
5188             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5189
5190             PlotXY (panel_handle, Panel_Graph, step, P21, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5191             PlotXY (panel_handle, Panel_Graph, step, TP2, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,
     1, VAL_WHITE);
5192             break;
5193         case 2: // Pressure [Pump #3 and TP-2]
5194             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5195
5196             PlotXY (panel_handle, Panel_Graph, step, P21, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5197             PlotXY (panel_handle, Panel_Graph, step, TP2, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,
     1, VAL_WHITE);
5198             break;
5199
5200         case 3: // Mass Flow Rate M#1
5201             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5202
5203             PlotXY (panel_handle, Panel_Graph, step, M13, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5204             break;
5205
5206         case 4: // Mass Flow Rate M#2
5207             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5208
5209             PlotXY (panel_handle, Panel_Graph, step, M23, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5210             break;
5211
5212         case 5: // Mass Flow Rate M#3
5213             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5214
5215             PlotXY (panel_handle, Panel_Graph, step, M33, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5216             break;
5217
5218         case 6: // Mass Flow Rate M#4
5219             DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5220
5221             PlotXY (panel handle, Panel Graph, step, M43, NUM, VAL DOUBLE, VAL DOUBLE, VAL SCATTER, VAL SOLID CIRCLE, VAL SOLID,
```

```
1, VAL_RED);
5222            break;
5223
5224        case 7: // Density M#1 Water
5225            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5226
5227            PlotXY (panel_handle, Panel_Graph, step, M11, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5228            break;
5229
5230        case 8: // Density M#2 Oil
5231            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5232
5233            PlotXY (panel_handle, Panel_Graph, step, M21, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5234            break;
5235
5236        case 9: // Density M#3 Emulsion
5237            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5238
5239            PlotXY (panel_handle, Panel_Graph, step, M31, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5240            break;
5241
5242        case 10:// Density M#4 Mixture out
5243            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5244
5245            PlotXY (panel_handle, Panel_Graph, step, M41, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5246            break;
5247
5248        case 11:// Pressure Inlet TP-2
5249            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5250
5251            PlotXY (panel_handle, Panel_Graph, step, TP2, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5252            break;
5253
5254        case 12:// Differential pressure dP1
5255            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5256
5257            PlotXY (panel_handle, Panel_Graph, step, DP1, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5258            break;
5259
5260        case 13://Delta pressure TP-2 - BRP-1
5261            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5262
5263            PlotXY (panel_handle, Panel_Graph, step, Delta, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID
, 1, VAL_RED);
5264
5265            break;
5266        case 14://Delta pressure TP-2 - BRP-1 vs dP1
5267            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5268
5269            PlotXY (panel_handle, Panel_Graph, step, Delta, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID
, 1, VAL_RED);
5270            PlotXY (panel_handle, Panel_Graph, step, DP1, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_FAT_LINE, VAL_EMPTY_SQUARE, VAL_SOLID,
 1, VAL_WHITE);
5271
5272            break;
5273
5274        case 15://Water injected
5275            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5276
5277            PlotXY (panel_handle, Panel_Graph, step, WATERINJ, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SO
LID, 1, VAL_RED);
5278            break;
5279
5280        case 16://OIL injected
5281            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5282
5283            PlotXY (panel_handle, Panel_Graph, step, OILINJ, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLI
D, 1, VAL_RED);
5284            break;
5285
5286        case 17://Emulsion injected
5287            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5288
5289            PlotXY (panel_handle, Panel_Graph, step, EMULINJ, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOL
ID, 1, VAL_RED);
5290            break;
5291
5292        case 18://  Temperature M#1 Water
5293            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5294
5295            PlotXY (panel_handle, Panel_Graph, step, M12, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_RED);
5296            PlotXY (panel_handle, Panel_Graph, step, M22, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_YELLOW);
5297            PlotXY (panel_handle, Panel_Graph, step, M32, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_DK_YELLOW);
5298            PlotXY (panel_handle, Panel_Graph, step, M42, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
1, VAL_DK_MAGENTA);
5299            break;
5300
5301        case 19://  Water flooding
```

```
5302            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5303
5304            PlotXY (panel_handle, Panel_Graph, step, WATERINJ2, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_S
    OLID, 1, VAL_RED);
5305            break;
5306
5307        case 20://  Water flooding  PV
5308            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5309
5310            PlotXY (panel_handle, Panel_Graph, step, WATERINJ2PV, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL
    _SOLID, 1, VAL_RED);
5311            break;
5312
5313        case 21:// Oil recovered vs Water injected PV
5314            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5315
5316            PlotXY (panel_handle, Panel_Graph, WATERINJ2PV, OILREC, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, V
    AL_SOLID, 1, VAL_RED);
5317            break;
5318
5319        case 22:// Oil recovered vs Time
5320            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5321
5322            PlotXY (panel_handle, Panel_Graph, step, OILREC, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLI
    D, 1, VAL_RED);
5323            break;
5324
5325        case 23://Swi vs Time
5326            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5327
5328            PlotXY (panel_handle, Panel_Graph, step, SWI, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5329            break;
5330
5331        case 24://Oil out at stage 9
5332            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5333
5334            PlotXY (panel_handle, Panel_Graph, step, OILREC_ST9, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_
    SOLID, 1, VAL_RED);
5335            break;
5336
5337        case 25://Water out at stage 9
5338
5339            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5340
5341            PlotXY (panel_handle, Panel_Graph, step, WATERREC_ST9, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VA
    L_SOLID, 1, VAL_RED);
5342            break;
5343
5344
5345        case 26://  Water inj PV vs Pressure drop
5346            DeleteGraphPlot (extra_g_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5347
5348            PlotXY (extra_g_handle, Panel_Graph, WATERINJ2PV, DP1, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VA
    L_SOLID, 1, VAL_RED);
5349
5350            break;
5351
5352        case 27://  Water inj PV vs Ref. pressure drop
5353            DeleteGraphPlot (extra_g_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5354
5355            PlotXY (extra_g_handle, Panel_Graph, WATERINJ2PV, Delta, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE,
    VAL_SOLID, 1, VAL_RED);
5356
5357            break;
5358
5359        case 28://  Water inj PV vs Ref. pressure drop vs DP1
5360            DeleteGraphPlot (extra_g_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5361
5362            PlotXY (extra_g_handle, Panel_Graph, WATERINJ2PV, Delta, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE,
    VAL_SOLID, 1, VAL_RED);
5363            PlotXY (extra_g_handle, Panel_Graph, WATERINJ2PV, DP1, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VA
    L_SOLID, 1, VAL_WHITE);
5364
5365            break;
5366
5367
5368        case 29:// Experimental stage [option underconstruction]
5369            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5370
5371            break;
5372
5373        case 30:// Only for GRAPH #6
5374            DeleteGraphPlot (panel_handle, Panel_Graph, -1,VAL_IMMEDIATE_DRAW);
5375
5376            PlotXY (panel_handle, Panel_Graph, step, M41, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL_SOLID,
    1, VAL_RED);
5377            PlotXY (panel_handle, Panel_Graph, step, SETPOINTOUT, NUM, VAL_DOUBLE, VAL_DOUBLE, VAL_SCATTER, VAL_SOLID_CIRCLE, VAL
    _SOLID, 1, VAL_RED);
5378
5379            break;
5380
5381    }
5382 return;
5383 }
5384
5385 int CVICALLBACK LOG ON OFF (int panel, int control, int event,
```

```
5386         void *callbackData, int eventData1, int eventData2)
5387 {
5388     switch (event)
5389         {
5390         case EVENT_COMMIT:
5391     // COM Ports
5392     GetCtrlVal(panel_handle, MAIN_LOG_P1,           &Is_P1);
5393                 if (p1check == 0) SetCtrlVal(panel_handle, MAIN_LOG_P1, 0);
5394
5395     GetCtrlVal(panel_handle, MAIN_LOG_P2,           &Is_P2);
5396                 if (p2check == 0) SetCtrlVal(panel_handle, MAIN_LOG_P2, 0);
5397
5398     GetCtrlVal(panel_handle, MAIN_LOG_P3,           &Is_P3);
5399                 if (p3check == 0) SetCtrlVal(panel_handle, MAIN_LOG_P3, 0);
5400     // Mass Flow Meters
5401     GetCtrlVal(panel_handle, MAIN_LOG_M1,           &Is_M1);
5402     GetCtrlVal(panel_handle, MAIN_LOG_M2,           &Is_M2);
5403     GetCtrlVal(panel_handle, MAIN_LOG_M3,           &Is_M3);
5404     GetCtrlVal(panel_handle, MAIN_LOG_M4,           &Is_M4);
5405     // Back Pressure Regulator
5406     GetCtrlVal(panel_handle, MAIN_LOG_BPR,          &Is_BPR);
5407     // Main Parameters
5408     GetCtrlVal(panel_handle, MAIN_LOG_Graph,        &Is_Graph);
5409     GetCtrlVal(panel_handle, MAIN_LOG_LOGtoFile,    &Is_Log);
5410     // DAQ
5411     GetCtrlVal(panel_handle, MAIN_LOG_DAQ,          &Is_DAQ);
5412     // Camera
5413     GetCtrlVal(panel_handle, MAIN_LOG_CAM,          &Is_Cam);
5414
5415     Update_Graphics();      // Update On-screen Graphics
5416             break;
5417
5418         case EVENT_RIGHT_CLICK:
5419             break;
5420             }
5421 return 0;
5422 }
5423
5424 void  Update_Graphics(void)
5425 {
5426 //  double  zero = 0;
5427     int Experiment_On;
5428
5429     GetCtrlVal(panel_handle, MAIN_START_STOP, &Experiment_On);  // Check to see if experiment is running
5430
5431 if(Experiment_On)   // Experiment is running
5432 {
5433
5434         EnableMassTurnControls(0);
5435
5436         if (Is_M1) EnableM1PanelControls(0);
5437                 else  EnableM1PanelControls(1);
5438
5439         if (Is_M2) EnableM2PanelControls(0);
5440                 else  EnableM2PanelControls(1);
5441
5442         if (Is_M3) EnableM3PanelControls(0);
5443                 else  EnableM3PanelControls(1);
5444
5445         if (Is_M4) EnableM4PanelControls(0);
5446                 else  EnableM4PanelControls(1);
5447
5448         if (Is_BPR) EnableBPRPanelControls(0);
5449                 else  EnableBPRPanelControls(1);
5450
5451             SetCtrlAttribute (panel_handle, MAIN_P1_PURGE, ATTR_DIMMED, 0);
5452             SetCtrlAttribute (panel_handle, MAIN_P1_RESTART, ATTR_DIMMED, 0);
5453             SetCtrlAttribute (panel_handle, MAIN_P1_5ERR, ATTR_DIMMED, 0);
5454
5455
5456
5457
5458             if (STG_1)  // check system
5459             {
5460                  SetCtrlAttribute (panel_handle, MAIN_STG_2, ATTR_DIMMED, 0);
5461                  SetCtrlAttribute (panel_handle, MAIN_WATER_INJ, ATTR_DIMMED, 0);
5462                  SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 8.5);
5463             }
5464
5465             if (STG_2)  // once confirmed - ask for water volume
5466             {
5467                  SetCtrlAttribute (panel_handle, MAIN_STG_3, ATTR_DIMMED, 0);
5468                  SetCtrlAttribute (panel_handle, MAIN_WATER_PV_INJ, ATTR_DIMMED, 0);
5469                  SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 17.0);
5470             }
5471
5472
5473             if (STG_3)  // waterflooding
5474             {
5475                  SetCtrlAttribute (panel_handle, MAIN_STG_4, ATTR_DIMMED, 0);
5476                  SetCtrlAttribute (panel_handle, MAIN_OIL_PV_INJ, ATTR_DIMMED, 0);
5477                  SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 25.5);
5478             }
5479
5480             if (STG_4) // Stop water flooding - Start oil flooding
5481             {
5482                  SetCtrlAttribute (panel handle, MAIN STG 5, ATTR DIMMED, 0);
```

```
5483                SetCtrlAttribute (panel_handle, MAIN_WATER_LEFT, ATTR_DIMMED, 0);
5484                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 34.00);
5485            }
5486
5487            if (STG_5) // No more water coming out
5488            {
5489                SetCtrlAttribute (panel_handle, MAIN_STG_6, ATTR_DIMMED, 0);
5490                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 42.50);
5491            }
5492
5493
5494            if (STG_6) // Select experiment
5495            {
5496                SetCtrlAttribute (panel_handle, MAIN_STG_7, ATTR_DIMMED, 0);
5497                SetCtrlAttribute (panel_handle, MAIN_WATER_PV_INJ_2, ATTR_DIMMED, 0);
5498                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 51.0);
5499
5500             GetCtrlVal(panel_handle, MAIN_SETEXP, &Type_exp);
5501
5502                    if (Type_exp == 2)     // Emulsion flooding
5503                        {
5504                            SetCtrlVal(panel_handle, MAIN_STG_7, 1);
5505                            SetCtrlVal(panel_handle, MAIN_STG_8, 1);
5506                            SetCtrlVal(panel_handle, MAIN_STG_9, 1);
5507                               STG_7 = 1;
5508                               STG_8 = 1;
5509                               STG_9 = 1;
5510                        }
5511                    else if (Type_exp == 3) //Alternate emulsion flooding
5512                        {
5513                            SetCtrlVal(panel_handle, MAIN_STG_7, 1);
5514                            SetCtrlVal(panel_handle, MAIN_STG_8, 1);
5515                            SetCtrlVal(panel_handle, MAIN_STG_9, 1);
5516                            SetCtrlVal(panel_handle, MAIN_STG_10, 1);
5517                                    STG_7 = 1;
5518                                    STG_8 = 1;
5519                                    STG_9 = 1;
5520                                    STG_10 = 1;
5521                        }
5522
5523            }
5524
5525            if (STG_7) // Waterflooding
5526            {
5527                SetCtrlAttribute (panel_handle, MAIN_STG_8, ATTR_DIMMED, 0);
5528                SetCtrlAttribute (panel_handle, MAIN_OIL_REC_PV, ATTR_DIMMED, 0);
5529                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 59.5);
5530
5531            }
5532
5533            if (STG_8) // No more oil comming out
5534            {
5535                SetCtrlAttribute (panel_handle, MAIN_STG_9, ATTR_DIMMED, 0);
5536                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 68.00);
5537            }
5538
5539            if (STG_9) // Stop water flooding - start emulsion flooding
5540            {
5541                SetCtrlAttribute (panel_handle, MAIN_STG_10, ATTR_DIMMED, 0);
5542                SetCtrlAttribute (panel_handle, MAIN_EMULSION_PV_INJ, ATTR_DIMMED, 0);
5543                SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 76.5);
5544            }
5545
5546            if (STG_10) // End for emulsion flooding
5547            {
5548                    SetCtrlAttribute (panel_handle, MAIN_STG_11, ATTR_DIMMED, 0);
5549
5550                    SetCtrlAttribute(panel_handle,MAIN_W_ALT_INJ, ATTR_DIMMED, 0);   //water injection     cc
5551                    SetCtrlAttribute(panel_handle,MAIN_WATER_PV_INJ_ALT, ATTR_DIMMED, 0);   //water injection  PV
5552                    SetCtrlAttribute(panel_handle,MAIN_E_ALT_INJ, ATTR_DIMMED, 0);   //emulsion injection  cc
5553                    SetCtrlAttribute(panel_handle,MAIN_EMULSION_PV_INJ_ALT, ATTR_DIMMED, 0);   //emulsion injection PV
5554
5555                    SetCtrlAttribute(panel_handle, MAIN_END_TEXT,ATTR_VISIBLE,1 );
5556                    SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 84.50);
5557            }
5558
5559            if (STG_11) // Alternate method
5560            {
5561                    SetCtrlAttribute (panel_handle, MAIN_STG_12, ATTR_DIMMED, 0);
5562                    SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 93.00);
5563            }
5564
5565            if (STG_12) //End of experiment
5566            {
5567                    MessagePopup("Info message","Experiment has been completed. Good luck with Data.");
5568                    SetCtrlVal (panel_handle, MAIN_EXP_PROGRESS, 100.00);
5569            }
5570
5571 }
5572 else    // Experiment is NOT running
5573 {
5574
5575            EnableMassTurnControls(1);
5576
5577         //  SetCtrlAttribute(panel_handle, MAIN_L_CORE,   ATTR_CTRL_MODE, VAL_HOT);
5578         //  SetCtrlAttribute(panel_handle, MAIN_D_CORE,   ATTR_CTRL_MODE, VAL_HOT);
5579         //  SetCtrlAttribute(panel_handle, MAIN_V_CORE,   ATTR_CTRL_MODE, VAL_HOT);
```

```
5580         //  SetCtrlAttribute(panel_handle, MAIN_POR_CORE, ATTR_CTRL_MODE, VAL_HOT);
5581
5582         // Dimmed M-1..4, BPR panels
5583
5584             EnableM1PanelControls(1);
5585             EnableM2PanelControls(1);
5586             EnableM3PanelControls(1);
5587             EnableM4PanelControls(1);
5588             EnableBPRPanelControls(1);
5589 }
5590 return;
5591 }
5592
5593 ////////////////////////////////////////////////////////////////////////
5594 ////////////////////////////////////////////////////////////////////////
5595
5596 int CVICALLBACK START_STOP (int panel, int control, int event,
5597         void *callbackData, int eventData1, int eventData2)
5598 { int test2;
5599   int i;  // ,j,delay;
5600
5601     switch (event)
5602     {
5603         case EVENT_COMMIT:
5604
5605            GetCtrlVal(panel_handle, MAIN_START_STOP, &test2);
5606
5607            SaveVars();
5608            Update_Graphics();
5609
5610            SetCtrlVal(panel_handle, MAIN_SAMPLES, SAMPLES);
5611
5612            if (!test2) // Stop experiment
5613            {
5614                forcestop = 1;
5615                Update_Graphics();
5616
5617                SetCtrlAttribute(panel_handle,MAIN_SNAP_SAVE,ATTR_DIMMED,1);
5618
5619             //--------------
5620             // STOP DAQ
5621             //--------------
5622                 if(Is_DAQ)
5623                 {
5624                     DAQquitflag = 0;
5625                     Pumps_quitflag=0;
5626                     Mass_quitflag=0;       //close Mass_Thread loop
5627                     BPR_quitflag=0;        //close BPR_Thread loop
5628                     Camera_quitflag=0;     //close Camera_Thread loop
5629                     Email_quitflag=0;      //close Email_Thread loop
5630
5631                     Delay(1);
5632
5633                     DAQmxStopTask(taskHandle);
5634                     DAQmxClearTask(taskHandle);
5635                     DAQ_Task_Started = 0;
5636
5637                 if( DAQdata )
5638                     free(DAQdata);
5639                 if( DAQArray_Out )
5640                     free(DAQArray_Out);
5641
5642                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, DAQthreadID);
5643                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, Mass_threadID);
5644                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, Pumps_threadID);
5645                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, BPR_threadID);
5646                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, Camera_threadID);
5647                     CmtReleaseThreadPoolFunctionID (DEFAULT_THREAD_POOL_HANDLE, Email_threadID);
5648
5649
5650                     Delay(1);
5651                 }
5652
5653              // Stop PUMPs #1-3
5654
5655                   STOPPUMPS();
5656
5657                   Delay(1);
5658
5659                   CmtDiscardLock (DAQ_lockHandle);
5660              }
5661              else // Start experiment
5662              {
5663
5664             forcestop = 0;
5665
5666             GeoMath();
5667             /*
5668             if (V_core == 0)
5669                       {
5670                       MessagePopup ("Warning","Geometry of sand pack is required.");
5671                       SetCtrlVal(panel_handle, MAIN_START_STOP, 0);
5672                       SetCtrlAttribute(panel_handle,MAIN_CORE_HOLDER,ATTR_FRAME_COLOR,VAL_RED);
5673                       break;
5674                       }
5675                  else    SetCtrlAttribute(panel_handle,MAIN_CORE_HOLDER,ATTR_FRAME_COLOR,VAL_LT_GRAY);
5676              */
```

```
5677
5678                //  Thread for Instruments, Logs, etc.
5679
5680                ThreadsRun();
5681
5682                SetCtrlAttribute(panel_handle,MAIN_SNAP_SAVE,ATTR_DIMMED,0);
5683
5684                // Undim Monitored Readouts and Adjust Control Types
5685
5686                Update_Graphics();
5687
5688
5689                //Enter into Experiment process
5690
5691                                stage = 1;
5692                                while (STG_1 != 1)
5693                                {
5694                                    ProcessSystemEvents ();
5695
5696                                    GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5697                                    BLINK();
5698                                    if (EM_Q==1 || forcestop==1)
5699                                    {
5700                                        goto Flag;
5701                                    }
5702                                    ProcessSystemEvents ();
5703                                }
5704
5705
5706                                DisplayPanel(sat_handle);
5707
5708                                stage = 2;
5709                                while (STG_2 !=1)
5710                                {
5711                                    ProcessSystemEvents ();
5712                                    GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5713                                    BLINK();
5714                                    if (EM_Q==1 || forcestop==1)
5715                                    {
5716                                        goto Flag;
5717                                    }
5718                                    ProcessSystemEvents ();
5719                                }
5720
5721                                stage = 3;
5722
5723                                CmtGetLock (Pumps_lockHandle);
5724                                    Delay (0.1);
5725
5726                                    wb_setpoint = 0.500;
5727                                    SetCtrlVal(panel_handle, MAIN_P1_SET_NFR, wb_setpoint);
5728
5729                                    SNFRP_1();
5730
5731                                    pumprun_1 = 11;    //save status about pump #1
5732
5733                                    panel_name=panel_handle;
5734                                    CP=P1_comport;
5735                                    send_data[0] = '\0';
5736                                    Fmt (send_data, "%s<ON\r");
5737                                    PTEXT_panel_name= MAIN_STATUS_P_1; //show response from pump
5738                                    SendReadPump ();
5739
5740                                CmtReleaseLock (Pumps_lockHandle);
5741
5742
5743                                while (STG_3 !=1)
5744                                {
5745                                    ProcessSystemEvents ();
5746
5747                                    SetCtrlVal(panel_handle,MAIN_WATER_PV_INJ,WATERINJPV[1]);  //ok //Show up value for the water
    flooding
5748
5749                                    GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5750                                    BLINK();
5751                                    if (EM_Q==1 || forcestop==1)
5752                                    {
5753                                        goto Flag;
5754                                    }
5755                                    ProcessSystemEvents ();
5756                                }
5757
5758                                if (STG_3 == 1)  STOPPUMPS();  //ok //stop ALL PUMPS [read as - waterflooding]
5759
5760                                stage = 4;              //stop waterflooding - start oil flooding
5761                                while (STG_4 !=1)
5762                                {
5763                                    ProcessSystemEvents ();
5764
5765                                    SetCtrlVal(panel_handle,MAIN_OIL_PV_INJ,OILINJPV[1]);    //ok
5766
5767                                    GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5768                                    BLINK();
5769                                    if (EM_Q==1 || forcestop==1)
5770                                    {
5771                                        goto Flag;
5772                                    }
```

```
5773                                ProcessSystemEvents ();
5774                              }
5775
5776
5777                              if (SWI[0] != 0.00)    SetCtrlVal (panel_handle, MAIN_WATER_LEFT, SWI[0]);     //set value to the
      panel.
5778                              else                   SetCtrlVal (panel_handle, MAIN_WATER_LEFT, SWI[1]);
5779
5780
5781                             stage = 5;
5782                             while (STG_5 !=1)
5783                             {
5784                                   ProcessSystemEvents ();
5785                                   BLINK();
5786                                   GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5787
5788                                   if (EM_Q==1 || forcestop==1)
5789                                   {
5790                                       goto Flag;
5791                                   }
5792                                   ProcessSystemEvents ();
5793                             }
5794
5795
5796                             if (STG_5 == 1) MessagePopup ("Warning","Please select flooding order.");
5797
5798                             stage = 6;
5799                             while (STG_6 !=1)  //ok //select experimental type
5800                             {
5801
5802                                   ProcessSystemEvents ();
5803                                   BLINK();
5804                                   GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5805
5806
5807                                   if (EM_Q==1 || forcestop==1)
5808                                   {
5809                                       goto Flag;
5810                                   }
5811                                   ProcessSystemEvents ();
5812                             }
5813
5814                            //start fraction collector
5815
5816                            SWITCH_FRACTION (1, 0, 0, 0);
5817                            Delay (0.5);
5818                            SWITCH_FRACTION (0, 0, 0, 0);
5819
5820                            stage = 7;
5821                            while (STG_7 != 1)    //waterflooding
5822                             {
5823                                   ProcessSystemEvents ();
5824                                   SetCtrlVal(panel_handle,MAIN_WATER_PV_INJ_2,WATERINJ2PV[1]);     //ok
5825
5826                                   BLINK();
5827                                   GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5828                                   if (EM_Q==1 || forcestop==1)
5829                                   {
5830                                       goto Flag;
5831                                   }
5832                                   ProcessSystemEvents ();
5833                             }
5834
5835                            if (STG_7 == 1)  STOPPUMPS();  //stop waterflooding
5836
5837
5838                             stage = 8;
5839                             while (STG_8 !=1)
5840                             {
5841                                   ProcessSystemEvents ();
5842                                   BLINK();
5843                                   GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5844                                   if (EM_Q==1 || forcestop==1)
5845                                   {
5846                                       goto Flag;
5847                                   }
5848                                   ProcessSystemEvents ();
5849                             }
5850
5851
5852                             stage = 9;
5853                             while (STG_9 !=1)
5854                             {
5855                                   ProcessSystemEvents ();
5856                                   BLINK();
5857
5858                                   GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
5859                                   if (EM_Q==1 || forcestop==1)
5860                                   {
5861                                       goto Flag;
5862                                   }
5863                                   ProcessSystemEvents ();
5864                             }
5865
5866
5867                             stage = 10;
5868                             while (STG 10 !=1)   //emulsion flooding
```

```
                                {
                                      ProcessSystemEvents ();
                                      SetCtrlVal(panel_handle,MAIN_EMULSION_PV_INJ,EMULINJPV[1]);
                                      BLINK();

                                      GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
                                      if (EM_Q==1 || forcestop==1)
                                      {
                                          goto Flag;
                                      }
                                      ProcessSystemEvents ();
                                }


                                if (STG_10 == 1)  STOPPUMPS();  //double check that all pumps are stopped

                                stage = 11;
                                while (STG_11 !=1)   //alternate flooding
                                {
                                      ProcessSystemEvents ();

                                      if (pumprun_1 == 11)
                                      {
                                          SetCtrlVal(panel_handle,MAIN_W_ALT_INJ,WATERINJ2[1]);
                                          SetCtrlVal(panel_handle,MAIN_WATER_PV_INJ_ALT,WATERINJ2PV[1]);
                                      }

                                      BLINK();

                                      GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
                                      if (EM_Q==1 || forcestop==1)
                                      {
                                          goto Flag;
                                      }
                                      ProcessSystemEvents ();
                                }


                                if (STG_11 == 1)  STOPPUMPS();  //double check that all pumps are stopped

                                stage = 12;
                                while (STG_12 !=1)    //done
                                {
                                      ProcessSystemEvents ();
                                      BLINK();


                                      GetCtrlVal (panel_handle, MAIN_EMERGENCY, &EM_Q);
                                      if (EM_Q==1 || forcestop==1)
                                      {
                                          goto Flag;
                                      }
                                      ProcessSystemEvents ();
                                }

Flag:

    if (EM_Q==1 || forcestop==1)
    {
    MessagePopup ("Warning","Experiment progress has been released.");
    //  EMG_STOP(1);

    }
          }

           break;
    }
    return 0;
}

int CVICALLBACK GEO_MATH (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            GeoMath();

            break;
    }
    return 0;
}

void GeoMath(void)
{int temp;

    GetCtrlVal(panel_handle, MAIN_2D_CH, &temp);

    if (temp == TRUE)
    {
        V_core = 2391.63 ; //2D configuration
        A_core = 20*2 ;   //2D configuration
    }
    else
    {
    GetCtrlVal(panel handle, MAIN L CORE, &L core);
```

```
5966     GetCtrlVal(panel_handle, MAIN_D_CORE, &D_core);
5967     A_core= pi*D_core*D_core / 4;
5968     V_core= A_core*L_core;    //bulk volume of rock
5969     }
5970     SetCtrlVal(panel_handle, MAIN_V_CORE, V_core);
5971
5972     // for permeability
5973         Perm_coef = 0.242 * L_core / A_core ;
5974
5975     // Porosity calculations, based on water saturation
5976     GetCtrlVal(panel_handle, MAIN_WATER_CORE, &WAT_core);
5977     Por_core=WAT_core/V_core;            //porosity = pore volume / bulk volume
5978     Por_core_per=Por_core*100;
5979     SetCtrlVal(panel_handle, MAIN_POR_CORE, Por_core_per);
5980
5981 return;
5982 }
5983
5984 int CVICALLBACK STG_NEXT (int panel, int control, int event,
5985         void *callbackData, int eventData1, int eventData2)
5986 {
5987     switch (event)
5988     {
5989         case EVENT_COMMIT:
5990
5991     // Equipment Setup
5992     GetCtrlVal(panel_handle, MAIN_STG_1,  &STG_1);
5993     GetCtrlVal(panel_handle, MAIN_STG_2,  &STG_2);
5994     GetCtrlVal(panel_handle, MAIN_STG_3,  &STG_3);
5995     GetCtrlVal(panel_handle, MAIN_STG_4,  &STG_4);
5996     GetCtrlVal(panel_handle, MAIN_STG_5,  &STG_5);
5997     GetCtrlVal(panel_handle, MAIN_STG_6,  &STG_6);
5998     GetCtrlVal(panel_handle, MAIN_STG_7,  &STG_7);
5999     GetCtrlVal(panel_handle, MAIN_STG_8,  &STG_8);
6000     GetCtrlVal(panel_handle, MAIN_STG_9,  &STG_9);
6001     GetCtrlVal(panel_handle, MAIN_STG_10,  &STG_10);
6002     GetCtrlVal(panel_handle, MAIN_STG_11,  &STG_11);
6003     GetCtrlVal(panel_handle, MAIN_STG_12,  &STG_12);
6004
6005     Update_Graphics();      // Update On-screen Graphics
6006
6007         break;
6008     }
6009     return 0;
6010 }
6011
6012 void EMG_STOP(int enable )
6013 {
6014
6015 SetCtrlAttribute(panel_handle, MAIN_STG_1, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6016 SetCtrlAttribute(panel_handle, MAIN_STG_2, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6017 SetCtrlAttribute(panel_handle, MAIN_STG_3, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6018 SetCtrlAttribute(panel_handle, MAIN_STG_4, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6019 SetCtrlAttribute(panel_handle, MAIN_STG_5, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6020 SetCtrlAttribute(panel_handle, MAIN_STG_6, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6021 SetCtrlAttribute(panel_handle, MAIN_STG_7, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6022 SetCtrlAttribute(panel_handle, MAIN_STG_8, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6023 SetCtrlAttribute(panel_handle, MAIN_STG_9, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6024 SetCtrlAttribute(panel_handle, MAIN_STG_10, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6025 SetCtrlAttribute(panel_handle, MAIN_STG_11, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6026 SetCtrlAttribute(panel_handle, MAIN_STG_12, ATTR_CTRL_MODE, VAL_INDICATOR,enable);
6027
6028 //SetPanelAttribute(panel_handle,ATTR_BACKCOLOR,VAL_RED);
6029
6030 return;
6031 }
6032
6033 void BLINK (void)
6034 {
6035    SetCtrlAttribute(panel_handle,MAIN_BLINK_TEXT,ATTR_TEXT_COLOR, VAL_WHITE);
6036    ProcessSystemEvents ();
6037    Delay (0.1);
6038    SetCtrlAttribute(panel_handle,MAIN_BLINK_TEXT,ATTR_TEXT_COLOR, VAL_GREEN);
6039
6040 return;
6041 }
6042
6043 void ThreadsRun (void)
6044 {
6045     // SETUP Pump's pressure monitoring system
6046
6047     //  Start a new thread function in the Default Thread Pool
6048         Pumps_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6049                           Pumps_ThreadFunction, NULL, &Pumps_threadID);
6050         Pumps_quitflag = 1;
6051         CmtNewLock (NULL, 0, &Pumps_lockHandle);
6052
6053     // SETUP Mass flow meters
6054     //  Start a new thread function for mass flow meters in the Default Thread Pool
6055         Mass_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6056                         Mass_ThreadFunction, NULL, &Mass_threadID);
6057         Mass_quitflag = 1;
6058         CmtNewLock (NULL, 0, &Mass_lockHandle);
6059
6060     // SETUP Backward pressure regulator
6061     // Start a new thread function for Back Pressure regulator in the Default Thread Pool
6062
```

```
6063         BPR_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6064                         BPR_ThreadFunction, NULL, &BPR_threadID);
6065         BPR_quitflag = 1;
6066         CmtNewLock (NULL, 0, &BPR_lockHandle);
6067 
6068     // SETUP thread for camera
6069         Camera_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6070                         Camera_ThreadFunction, NULL, &Camera_threadID);
6071         Camera_quitflag = 1;
6072         CmtNewLock (NULL, 0, &Camera_lockHandle);
6073 
6074     // SETUP thread for email
6075         Email_cmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6076                         Email_ThreadFunction, NULL, &Email_threadID);
6077         Email_quitflag = 1;
6078         CmtNewLock (NULL, 0, &Email_lockHandle);
6079 
6080 
6081     // SETUP DAQ
6082         if(Is_DAQ)
6083         {
6084             DSN_Setup_DAQ();
6085         //  Start a new thread function in the Default Thread Pool
6086             DAQcmtStatus = CmtScheduleThreadPoolFunction (DEFAULT_THREAD_POOL_HANDLE,
6087                         DAQThreadFunction, NULL, &DAQthreadID);
6088             DAQquitflag = 1;
6089             CmtNewLock (NULL, 0, &DAQ_lockHandle);
6090         }
6091 return;
6092 }
6093 
6094 void STOPPUMPS (void)
6095 {
6096 if (Pumps_quitflag == 1)
6097     {
6098     CmtGetLock (Pumps_lockHandle);
6099     Delay (0.1);
6100     STOPPUMPSsub();
6101     CmtReleaseLock (Pumps_lockHandle);
6102     }
6103 else STOPPUMPSsub();
6104 return;
6105 }
6106 
6107 void STOPPUMPSsub (void)
6108 {
6109     if (Is_P1)  //off pump #1
6110     {
6111     FlushOutQ (P1_comport);
6112     FlushInQ (P1_comport);
6113 
6114     send_data[0] = '\0';
6115     Fmt (send_data, "%s<OFF\r");
6116 
6117     stringsize = StringLength (send_data);
6118     bytes_sent = ComWrt (P1_comport, send_data, stringsize);
6119                                         Delay(0.05); // 0.05
6120 
6121     read_cnt = GetInQLen (P1_comport);
6122 
6123     read_data[0] = '\0';
6124     read_term = 13;
6125     bytes_read = ComRdTerm (P1_comport, read_data, read_cnt, read_term);
6126 
6127     CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
6128     SetCtrlVal (panel_handle, MAIN_STATUS_P_1, tbox_read_data);
6129 
6130     pumprun_1 = 12;
6131 
6132     //ERROR EVENT LIST
6133     SetCtrlAttribute(panel_handle,MAIN_STOP_P_1,ATTR_VISIBLE, 1);
6134     SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_3,ATTR_VISIBLE, 1);
6135 
6136     RS232Error = ReturnRS232Err ();
6137     if (RS232Error)
6138         DisplayRS232Error ();
6139     send_data[0] = '\0';
6140     }
6141             if (Is_P2)  //off pump #2
6142             {
6143             FlushOutQ (P2_comport);
6144             FlushInQ (P2_comport);
6145 
6146             send_data[0] = '\0';
6147             Fmt (send_data, "%s<OFF\r");
6148 
6149             stringsize = StringLength (send_data);
6150             bytes_sent = ComWrt (P2_comport, send_data, stringsize);
6151             Delay(0.05); // 0.05
6152 
6153                     read_cnt = GetInQLen (P2_comport);
6154 
6155                     read_data[0] = '\0';
6156                     read_term = 13;
6157                     bytes_read = ComRdTerm (P2_comport, read_data, read_cnt, read_term);
6158 
6159                     CopyString (tbox read data, 0, read data, 0, bytes read);
```

```
6160                    SetCtrlVal (panel_handle, MAIN_STATUS_P_2, tbox_read_data);
6161
6162                    pumprun_2 = 22;
6163
6164                    //ERROR EVENT LIST
6165                    SetCtrlAttribute(panel_handle,MAIN_STOP_P_2,ATTR_VISIBLE, 1);
6166                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_4,ATTR_VISIBLE, 1);
6167
6168                    RS232Error = ReturnRS232Err ();
6169                    if (RS232Error)
6170                        DisplayRS232Error ();
6171                    send_data[0] = '\0';
6172            }
6173                    if (Is_P3)  //off pump #3
6174                    {
6175                    FlushOutQ (P3_comport);
6176                    FlushInQ (P3_comport);
6177
6178                    send_data[0] = '\0';
6179                    Fmt (send_data, "%s<OFF\r");
6180
6181                    stringsize = StringLength (send_data);
6182                    bytes_sent = ComWrt (P3_comport, send_data, stringsize);
6183                    Delay(0.05); // 0.05
6184
6185                            read_cnt = GetInQLen (P3_comport);
6186
6187                            read_data[0] = '\0';
6188                            read_term = 13;
6189                            bytes_read = ComRdTerm (P3_comport, read_data, read_cnt, read_term);
6190
6191                            CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
6192                            SetCtrlVal (panel_handle, MAIN_STATUS_P_3, tbox_read_data);
6193
6194                            pumprun_3 = 32;
6195
6196                            //ERROR EVENT LIST
6197                            SetCtrlAttribute(panel_handle,MAIN_STOP_P_3,ATTR_VISIBLE, 1);
6198                            SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_5,ATTR_VISIBLE, 1);
6199
6200                            RS232Error = ReturnRS232Err ();
6201                            if (RS232Error)
6202                                DisplayRS232Error ();
6203                            send_data[0] = '\0';
6204                    }
6205 return;
6206 }
6207
6208 int CVICALLBACK ZeroErrorCallBack (int panel, int control, int event,
6209         void *callbackData, int eventData1, int eventData2)
6210 {
6211 double data;
6212
6213     switch (event)
6214     {
6215         case EVENT_COMMIT:
6216
6217             switch(control)
6218             {
6219                 case MAIN_ZERO_TP2:   //upstream
6220
6221                      GetCtrlVal(panel_handle,MAIN_TP2, &data);
6222
6223                      ZEROTP2[0]=ZEROTP2[0] + data;
6224                 break;
6225                 case MAIN_ZERO_DP1:
6226
6227                      GetCtrlVal(panel_handle,MAIN_DP1, &data);
6228                      ZERODP1[0]=ZERODP1[0] + data;
6229                 break;
6230                 case MAIN_ZERO_TP1:   //downstream
6231
6232                      GetCtrlVal(panel_handle,MAIN_TP1, &data);
6233                      ZEROTP1[0]=ZEROTP1[0] + data;
6234                 break;
6235
6236
6237                 case PRESSURE_ZERO_CP1_1:   // CP1_1
6238                      GetCtrlVal(pressure_handle,PRESSURE_CP1_1, &data);
6239                      ZEROCP11[0]=ZEROCP11[0] + data;
6240                 break;
6241                 case PRESSURE_ZERO_CP1_2:   // CP1_2
6242                      GetCtrlVal(pressure_handle,PRESSURE_CP1_2, &data);
6243                      ZEROCP12[0]=ZEROCP12[0] + data;
6244                 break;
6245                 case PRESSURE_ZERO_CP1_3:   // CP1_3
6246                      GetCtrlVal(pressure_handle,PRESSURE_CP1_3, &data);
6247                      ZEROCP13[0]=ZEROCP13[0] + data;
6248                 break;
6249                 case PRESSURE_ZERO_CP1_4:   // CP1_4
6250                      GetCtrlVal(pressure_handle,PRESSURE_CP1_4, &data);
6251                      ZEROCP14[0]=ZEROCP14[0] + data;
6252                 break;
6253                 case PRESSURE_ZERO_CP1_5:   // CP1_5
6254                      GetCtrlVal(pressure_handle,PRESSURE_CP1_5, &data);
6255                      ZEROCP15[0]=ZEROCP15[0] + data;
6256                 break;
```

```
6257
6258                case PRESSURE_ZERO_CP2_1:   // CP2_1
6259                     GetCtrlVal(pressure_handle,PRESSURE_CP2_1, &data);
6260                     ZEROCP21[0]=ZEROCP21[0] + data;
6261                break;
6262                case PRESSURE_ZERO_CP2_2:   // CP2_2
6263                     GetCtrlVal(pressure_handle,PRESSURE_CP2_2, &data);
6264                     ZEROCP22[0]=ZEROCP22[0] + data;
6265                break;
6266                case PRESSURE_ZERO_CP2_3:   // CP2_3
6267                     GetCtrlVal(pressure_handle,PRESSURE_CP2_3, &data);
6268                     ZEROCP23[0]=ZEROCP23[0] + data;
6269                break;
6270                case PRESSURE_ZERO_CP2_4:   // CP2_4
6271                     GetCtrlVal(pressure_handle,PRESSURE_CP2_4, &data);
6272                     ZEROCP24[0]=ZEROCP24[0] + data;
6273                break;
6274                case PRESSURE_ZERO_CP2_5:   // CP2_5
6275                     GetCtrlVal(pressure_handle,PRESSURE_CP2_5, &data);
6276                     ZEROCP25[0]=ZEROCP25[0] + data;
6277                break;
6278
6279                case PRESSURE_ZERO_CP3_1:   // CP3_1
6280                     GetCtrlVal(pressure_handle,PRESSURE_CP3_1, &data);
6281                     ZEROCP31[0]=ZEROCP31[0] + data;
6282                break;
6283                case PRESSURE_ZERO_CP3_2:   // CP3_2
6284                     GetCtrlVal(pressure_handle,PRESSURE_CP3_2, &data);
6285                     ZEROCP32[0]=ZEROCP32[0] + data;
6286                break;
6287                case PRESSURE_ZERO_CP3_3:   // CP3_3
6288                     GetCtrlVal(pressure_handle,PRESSURE_CP3_3, &data);
6289                     ZEROCP33[0]=ZEROCP33[0] + data;
6290                break;
6291                case PRESSURE_ZERO_CP3_4:   // CP3_4
6292                     GetCtrlVal(pressure_handle,PRESSURE_CP3_4, &data);
6293                     ZEROCP34[0]=ZEROCP34[0] + data;
6294                break;
6295                case PRESSURE_ZERO_CP3_5:   // CP3_5
6296                     GetCtrlVal(pressure_handle,PRESSURE_CP3_5, &data);
6297                     ZEROCP35[0]=ZEROCP35[0] + data;
6298                break;
6299            }
6300            break;
6301
6302        case EVENT_RIGHT_CLICK:
6303            break;
6304    }
6305    return 0;
6306 }
6307
6308 int CVICALLBACK ErrReleaseCallBack (int panel, int control, int event,
6309         void *callbackData, int eventData1, int eventData2)
6310 {
6311    switch (event)
6312    {
6313        case EVENT_COMMIT:
6314          switch(control)
6315          {
6316             case MAIN_ERROR_RELEASE_1:
6317                    SetCtrlAttribute(panel_handle,MAIN_DIFF_OVER_150,ATTR_VISIBLE, 0);
6318                    SetCtrlVal(panel_handle,MAIN_ERROR_RELEASE_3,0);
6319                    OVERPRES = 1;
6320                    break;
6321             case MAIN_ERROR_RELEASE_2:
6322                    SetCtrlAttribute(panel_handle,MAIN_TP2_OVER_1450,ATTR_VISIBLE, 0);
6323                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_2,ATTR_VISIBLE, 0);
6324                    break;
6325             case MAIN_ERROR_RELEASE_3:
6326                    SetCtrlAttribute(panel_handle,MAIN_STOP_P_1,ATTR_VISIBLE, 0);
6327                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_3,ATTR_VISIBLE, 0);
6328                    break;
6329             case MAIN_ERROR_RELEASE_4:
6330                    SetCtrlAttribute(panel_handle,MAIN_STOP_P_2,ATTR_VISIBLE, 0);
6331                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_4,ATTR_VISIBLE, 0);
6332                    break;
6333             case MAIN_ERROR_RELEASE_5:
6334                    SetCtrlAttribute(panel_handle,MAIN_STOP_P_3,ATTR_VISIBLE, 0);
6335                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_5,ATTR_VISIBLE, 0);
6336                    break;
6337             case MAIN_ERROR_RELEASE_6:
6338                    SetCtrlAttribute(panel_handle,MAIN_ERR_BPR_ALARM,ATTR_VISIBLE, 0);
6339                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_6,ATTR_VISIBLE, 0);
6340                    break;
6341             case MAIN_ERROR_RELEASE_7:
6342                    SetCtrlAttribute(panel_handle,MAIN_TP1_OVER_1450,ATTR_VISIBLE, 0);
6343                    SetCtrlAttribute(panel_handle,MAIN_ERROR_RELEASE_7,ATTR_VISIBLE, 0);
6344                    break;
6345          }
6346        break;
6347    }
6348    return 0;
6349 }
6350
6351 int CVICALLBACK UnitCallBack (int panel, int control, int event,
6352         void *callbackData, int eventData1, int eventData2)
6353 {
```

```
int visunit, permunit;

    switch (event)
    {
        case EVENT_COMMIT:
             vispermunit();
             SaveConfig(0);
        break;
    }
    return 0;
}

void vispermunit(void)
{
int visunit, permunit;
GetCtrlVal(panel_handle,MAIN_VIS_UNIT, &visunit);
GetCtrlVal(panel_handle,MAIN_PERM_UNIT, &permunit);

                 if (visunit == 0)  viscoef=  0.001 ;
                    else if (visunit == 1) viscoef= 1;
                        else  viscoef= 1;

                 if (permunit == 0)  permcoef= 1;
                        else  permcoef= 1000;
return;
}

int CVICALLBACK CleanoutCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
double temp;

    switch (event)
    {
        case EVENT_COMMIT:
          switch(control)
          {
             case MAIN_SET_OUT:

                GetCtrlVal (panel_handle, MAIN_UPLIM_OUT, &temp);
                SetCtrlAttribute (panel_handle, MAIN_OUTLETTANK,ATTR_MAX_VALUE, temp);

             break;
             case MAIN_SET_AND_CLEAN_OUT:

                GetCtrlVal (panel_handle, MAIN_UPLIM_OUT, &temp);
                SetCtrlAttribute (panel_handle, MAIN_OUTLETTANK,ATTR_MAX_VALUE, temp);

                OUTLET[1]=0.0;
                OUTLET[0]=0.0;
                SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK

             break;
             case MAIN_CLEAN_OUT:

                OUTLET[1]=0.0;
                OUTLET[0]=0.0;
                SetCtrlVal (panel_handle, MAIN_OUTLETTANK, OUTLET[0]);  //set new value to TANK

             break;
          }
        break;
    }
    return 0;
}

int CVICALLBACK OilWaterCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
double temp;

    switch (event)
    {
        case EVENT_COMMIT:
            SetCtrlVal (panel_handle,  MAIN_LED_NEWSET_PLOT_CHECK,   1);
            Delay (0.1);
            GetCtrlVal (panel_handle, MAIN_OILWATER_SET, &temp);

            SETPOINTOUT[0]=temp;

            SetCtrlVal (panel_handle,  MAIN_LED_NEWSET_PLOT_CHECK,   0);

            break;
    }
    return 0;
}

int CVICALLBACK WaterinjCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:

            GetCtrlVal (sat_handle, SAT_WATER_INJ, &PV[0]);
            SetCtrlVal (panel_handle, MAIN_STG_2, 1);

```

```
6451             GetCtrlVal(panel_handle, MAIN_STG_2,  &STG_2);
6452             Update_Graphics();
6453
6454
6455             SetCtrlVal(panel_handle,MAIN_WATER_INJ,PV[0]);   //
6456             SetCtrlVal(panel_handle,MAIN_WATER_CORE,PV[0]);  //
6457
6458
6459             GeoMath();
6460
6461             Delay(0.1);
6462
6463             DiscardPanel (sat_handle);
6464
6465             break;
6466     }
6467     return 0;
6468 }
6469
6470
6471 void SWITCH_FRACTION (int channel1, int channel2, int channel3, int channel4)      //Function to switch solenoid valves
6472 {
6473 int         error=0;
6474 TaskHandle  taskHandleAV2=0;
6475 char        chanAV2[256];
6476 uInt8       dataAV2[4];
6477 char        errBuffAV2[2048]={'\0'};
6478
6479
6480         GetCtrlVal(config_handle,CONFIG_SOLCHANNEL_2,chanAV2);
6481
6482
6483         dataAV2[0] = channel1; // green run   [start]
6484         dataAV2[1] = channel2; // red run     [test tube]
6485         dataAV2[2] = channel3; // white run   [stop]
6486         dataAV2[3] = channel4; // restart run [restart]
6487
6488         if  (dataAV2[0] == 1) SetCtrlVal (panel_handle, MAIN_LED_FC, 1);  //Show up status of Fraction collector
6489
6490         /*********************************************/
6491         // DAQmx Configure Code
6492         /*********************************************/
6493         SetWaitCursor(1);
6494         DAQmxErrChk (DAQmxCreateTask("",&taskHandleAV2));
6495         DAQmxErrChk (DAQmxCreateDOChan(taskHandleAV2,chanAV2,"",DAQmx_Val_ChanForAllLines));
6496
6497         /*********************************************/
6498         // DAQmx Start Code
6499         /*********************************************/
6500         DAQmxErrChk (DAQmxStartTask(taskHandleAV2));
6501
6502         /*********************************************/
6503         // DAQmx Write Code
6504         /*********************************************/
6505         DAQmxErrChk (DAQmxWriteDigitalLines(taskHandleAV2,1,1,10.0,DAQmx_Val_GroupByChannel,dataAV2,NULL,NULL));
6506
6507
6508 Error:
6509     SetWaitCursor(0);
6510     if( DAQmxFailed(error) )
6511         DAQmxGetExtendedErrorInfo(errBuffAV2,2048);
6512     if( taskHandleAV2!=0 ) {
6513         /*********************************************/
6514         // DAQmx Stop Code
6515         /*********************************************/
6516         DAQmxStopTask(taskHandleAV2);
6517         DAQmxClearTask(taskHandleAV2);
6518     }
6519     if( DAQmxFailed(error) )
6520         MessagePopup("DAQmx Error",errBuffAV2);
6521
6522 return;
6523 }
6524
6525 int CVICALLBACK ConnectCallBack (int panel, int control, int event,
6526         void *callbackData, int eventData1, int eventData2)
6527 {
6528 int temp;
6529     switch (event)
6530     {
6531         case EVENT_COMMIT:
6532              GetCtrlVal(config_handle, CONFIG_CONNECT, &temp);
6533              if (temp == TRUE) DSN_Init_Camera();
6534              else DeInit_Camera();
6535         break;
6536     }
6537     return 0;
6538 }
6539
6540 // DSN_Init_Image           : Init the Image position
6541
6542 void DSN_Init_Image(int panelHandle)
6543 {
6544     Point windowPos;
6545     int firstWindowWidth;
6546     int mainPanelTop;
6547     int mainPanelHeight;
```

```
6548     int mainPanelWidth;
6549     int mainPanelLeft;
6550
6551     int Mon_height;
6552     int Mon_width;
6553     int MONITOR_ID;
6554     int im_width;
6555     int im_height;
6556     int imWin_width, imWin_heigth;
6557
6558     GetMonitorAttribute (1, ATTR_NEXT_MONITOR, &MONITOR_ID);
6559
6560     // Get the location of the top of the panel
6561     GetPanelAttribute( panelHandle, ATTR_TOP, &mainPanelTop );
6562     // Get the location of the left side of the panel
6563     GetPanelAttribute( panelHandle, ATTR_LEFT, &mainPanelLeft );
6564     // Get the height of the panel
6565     GetPanelAttribute( panelHandle, ATTR_HEIGHT, &mainPanelHeight );
6566     // Get the height of the panel
6567     GetPanelAttribute (panelHandle, ATTR_WIDTH, &mainPanelWidth);
6568     // Set up windowPos
6569     windowPos.x = mainPanelLeft + mainPanelWidth;
6570     windowPos.y = mainPanelTop - 25;
6571
6572     imaqMoveWindow( SOURCE_WINDOW, windowPos );
6573
6574     // Reduces size of the imaqWindow
6575     GetCtrlVal(panelHandle, PANEL_im_heigth, &im_height);
6576     GetCtrlVal(panelHandle, PANEL_im_width, &im_width);
6577
6578     imaqSetWindowSize      (SOURCE_WINDOW, im_height , im_width);
6579     imaqSetWindowZoomToFit (SOURCE_WINDOW, TRUE);
6580
6581     GetMonitorAttribute (1, ATTR_HEIGHT, &Mon_height);
6582     GetMonitorAttribute (1, ATTR_WIDTH,  &Mon_width);
6583     SetCtrlVal(panelHandle, PANEL_Mon_heigth, Mon_height);
6584     SetCtrlVal(panelHandle, PANEL_Mon_width,  Mon_width);
6585
6586     // Set the size of the image window
6587     imWin_width =  Mon_width - mainPanelWidth - mainPanelLeft;
6588     imWin_heigth = Mon_height - 25;
6589 return;
6590 }
6591
6592 // Convert the error to a string and present to user
6593 void DisplayError(IMAQdxError error)
6594 {
6595     char errorString[256];
6596     IMAQdxGetErrorString(error, errorString, sizeof(errorString));
6597     MessagePopup("Error", errorString);
6598 }
6599
6600 // DSN_Init_Camera: Init the Camera: Get information
6601 void DSN_Init_Camera(void)
6602 {
6603     unsigned int    attributeCount;
6604     int             i=0;
6605
6606
6607     IMAQdxError     error2 = IMAQdxErrorSuccess;
6608
6609 // CAMERA INITIALIZATION
6610         // Get the interface name
6611         GetCtrlVal (config_handle, CONFIG_CAMNAME, camName);
6612
6613
6614         SetCtrlAttribute(config_handle,CONFIG_OpenSet,ATTR_VISIBLE,1 );
6615
6616         // Dispose any images that were previously created
6617         if(images)
6618         {
6619             for(i=0; i<imageArraySize; i++)
6620                 imaqDispose (images[i]);
6621                 free(images);
6622                 images = NULL;
6623         }
6624         // Open a session on the selected camera
6625         error2 = IMAQdxOpenCamera (camName, IMAQdxCameraControlModeController, &session);
6626         // Return if we did not open the camera
6627         if(error2)
6628         {
6629             DisplayError(error2);
6630             //return 0;
6631         }
6632
6633 //  Set up the CAMERA ATTRIBUTES DIALOG
6634
6635 // Enumerate the attributes supported by the camera
6636
6637             ClearListCtrl (panelHandle, PANEL_CAM_ATTR);
6638             error2 = IMAQdxEnumerateAttributes (session, NULL, &attributeCount, NULL);
6639             if (error2)
6640             {
6641             //  break;
6642             }
6643             cameraAttributes = (IMAQdxAttributeInformation *)malloc(attributeCount * sizeof(IMAQdxAttributeInformation));
6644             error2 = IMAQdxEnumerateAttributes (session, cameraAttributes, &attributeCount, NULL);
```

```
6645             if (error2)
6646             {
6647             //  break;
6648             }
6649
6650 //Populate the available attributes into the table control
6651
6652             for (i=0; i < attributeCount; i++)
6653             {
6654                 InsertListItem (panelHandle, PANEL_CAM_ATTR, -1, cameraAttributes[i].Name, i);
6655             }
6656
6657 // Display the details for the current attribute
6658
6659             UpdateAttributeTab();
6660
6661 // Turn the GRAB on
6662
6663         SetCtrlAttribute (panel_handle, MAIN_SNAP,      ATTR_DIMMED, 0);
6664 //      SetCtrlAttribute (panel_handle, MAIN_SNAP_SAVE, ATTR_DIMMED, 0);
6665         SetCtrlAttribute (panel_handle, MAIN_GRAB_2,    ATTR_DIMMED, 0);
6666 }
6667
6668  void DeInit_Camera(void)
6669 {
6670     unsigned int    attributeCount;
6671     int             i=0;
6672     IMAQdxError     error2 = IMAQdxErrorSuccess;
6673
6674 // Camera De-initialization
6675
6676         if(images)
6677         {
6678             for(i=0; i<imageArraySize; i++)
6679                 imaqDispose (images[i]);
6680                 free(images);
6681                 images = NULL;
6682         }
6683
6684
6685         SetCtrlAttribute (panel_handle, MAIN_SNAP, ATTR_DIMMED,      1);
6686     //  SetCtrlAttribute (panel_handle, MAIN_SNAP_SAVE, ATTR_DIMMED, 1);
6687         SetCtrlAttribute (panel_handle, MAIN_GRAB_2, ATTR_DIMMED,    1);
6688
6689         error2 = IMAQdxCloseCamera (session);
6690
6691         // Return if we did not close the camera
6692         if(error2)
6693         {
6694             DisplayError(error2);
6695         }
6696
6697 //  Diable up the CAMERA ATTRIBUTES DIALOG
6698 SetCtrlAttribute(config_handle,CONFIG_OpenSet,ATTR_VISIBLE, 0);
6699 }
6700
6701 int CVICALLBACK OpenSetCallBack (int panel, int control, int event,
6702         void *callbackData, int eventData1, int eventData2)
6703 {
6704     switch (event)
6705     {
6706         case EVENT_COMMIT:
6707             DisplayPanel(panelHandle);
6708             break;
6709     }
6710     return 0;
6711 }
6712
6713 int CVICALLBACK CloseSetCallBack (int panel, int control, int event,
6714         void *callbackData, int eventData1, int eventData2)
6715 {
6716     switch (event)
6717     {
6718         case EVENT_COMMIT:
6719             HidePanel(panelHandle);
6720             break;
6721     }
6722     return 0;
6723 }
6724
6725 int CVICALLBACK Start_Grab (int panel, int control, int event,
6726         void *callbackData, int eventData1, int eventData2)
6727 {
6728 int test,i;
6729 double time;
6730
6731     switch (event)
6732     {
6733         case EVENT_COMMIT:
6734
6735             IMAQdxUnconfigureAcquisition(session);
6736             IMAQdxConfigureGrab (session);
6737
6738             GetCtrlVal (panel_handle, MAIN_GRAB_2, &test);
6739             if(test)
6740             {
6741                 // Create an image
```

```
6742                 if(image_g)
6743                     imaqDispose(image_g);
6744                 image_g = imaqCreateImage (IMAQ_IMAGE_U8, 0);
6745                 // Configure the acquisiton
6746                 IMAQdxConfigureAcquisition (session, TRUE, 3);
6747                 // Start the acquisition
6748                 IMAQdxStartAcquisition (session);
6749                 // RUN the timer to do the GRAB and DISPLAY
6750                 SetCtrlAttribute (panel_handle, MAIN_TIMER, ATTR_ENABLED, TRUE);
6751             }
6752             else
6753             {
6754                 // Stop the Imaq Loop
6755                 SetCtrlAttribute (panel_handle, MAIN_TIMER, ATTR_ENABLED, FALSE);
6756                 // Stop the acquisition
6757                 IMAQdxStopAcquisition (session);
6758                 // Unconfigure the acquisition
6759                 IMAQdxUnconfigureAcquisition (session);
6760 
6761                 if(image_g)
6762                 {
6763                     imaqDispose(image_g);
6764                     image_g = NULL;
6765                 }
6766             }
6767             break;
6768     }
6769     return 0;
6770 }
6771 
6772 int CVICALLBACK Imaq_loop (int panel, int control, int event,
6773         void *callbackData, int eventData1, int eventData2)
6774 {
6775 
6776     const double frameRateInterval = 0.5;
6777     static double lastTick = 0;
6778     double newTick;
6779     static unsigned int lastBufferNumber = - 1;
6780     unsigned int bufferNumber;
6781     unsigned int temp;
6782     IMAQdxError error2 = IMAQdxErrorSuccess;
6783 
6784     switch (event)
6785     {
6786         case EVENT_TIMER_TICK:
6787             newTick = Timer();
6788 
6789             // Get the next frame
6790             error2 = IMAQdxGrab (session, image_g, TRUE, &bufferNumber);
6791             if (error2)
6792             {
6793                 break;
6794             }
6795 
6796             // Display using NI Vision
6797             imaqDisplayImage (image_g, 0, TRUE);
6798 
6799             // Set settings to the NI Vision window
6800             if (Just_Grabbed == TRUE)
6801             {
6802                 DSN_Init_Image(panelHandle);
6803                 Just_Grabbed = 0;
6804             }
6805 
6806             // Calculate the number of frame per seconds
6807             if ((newTick - lastTick) >= frameRateInterval)
6808             {
6809                 double frameRate = (double)(bufferNumber - lastBufferNumber) / (newTick - lastTick);
6810                 lastTick = newTick;
6811                 lastBufferNumber = bufferNumber;
6812             }
6813             break;
6814     }
6815 
6816     if (error2) { DisplayError(error2); }
6817 
6818 return 0;
6819 }
6820 
6821 
6822 // UpdateAttributeTab   : Load and display the ATTRIBUTES
6823 void UpdateAttributeTab(void) {
6824     char            Units[IMAQDX_MAX_API_STRING_LENGTH];
6825     char            Tooltip[IMAQDX_MAX_API_STRING_LENGTH];
6826     unsigned int    U32minimum = 0;
6827     unsigned int    U32maximum = 1;
6828     unsigned int    U32increment = 1;
6829     double          DBLminimum = 0;
6830     double          DBLmaximum = 1;
6831     double          DBLincrement = 0;
6832     BOOL            Readable = 0;
6833     BOOL            Writable = 0;
6834     int             i = 0;
6835     unsigned int    U32current = 0;
6836     double          DBLcurrent = 0;
6837     char            STRcurrent[IMAQDX_MAX_API_STRING_LENGTH];
6838     unsigned int    EnumCurrent = 0;
```

```
6839     BOOL            BOOLcurrent = FALSE;
6840
6841     SetCtrlVal(panelHandle, PANEL_LED_EXP_7, 1);
6842
6843     // Get the selected attribute, and make the tab control visible
6844     GetCtrlVal (panelHandle, PANEL_CAM_ATTR, &selectedAttrIndex);
6845     SetCtrlVal (panelHandle, PANEL_Cam_List_Num, selectedAttrIndex);
6846     SetCtrlAttribute (panelHandle, PANEL_CAM_TAB, ATTR_VISIBLE, TRUE);
6847
6848     // Get the units for the attribute and the tooltip
6849     IMAQdxGetAttributeUnits (session, cameraAttributes[selectedAttrIndex].Name, Units, IMAQDX_MAX_API_STRING_LENGTH);
6850     SetCtrlVal (panelHandle, PANEL_UNITS, Units);
6851     IMAQdxGetAttributeTooltip (session, cameraAttributes[selectedAttrIndex].Name, Tooltip, IMAQDX_MAX_API_STRING_LENGTH);
6852     ResetTextBox (panelHandle, PANEL_TOOLTIPS, Tooltip);
6853     SetCtrlAttribute (panelHandle, PANEL_TOOLTIPS, ATTR_WRAP_MODE, VAL_WORD_WRAP);
6854
6855     // Determine the data type for the attribute, display associated tab
6856     // Determine if the attribute is writable, readable, or neither, and dim/hide control accordingly
6857     // For U32, get minimum, maximum, increment, and current value from driver, and configure the control
6858     if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeU32)
6859     {
6860         U32current = 0;
6861         SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 0);
6862         GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 0, &TabPanelHandle);
6863         IMAQdxGetAttributeMinimum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, &U32minimum);
6864         IMAQdxGetAttributeMaximum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, &U32maximum);
6865         IMAQdxGetAttributeIncrement (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, &U32increment);
6866         IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
6867         IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
6868         if (Readable == TRUE)
6869             IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, &U32current);
6870         SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_MIN_VALUE, U32minimum);
6871         SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_MAX_VALUE, U32maximum);
6872         SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_INCR_VALUE, U32increment);
6873         SetCtrlVal (TabPanelHandle, U32TAB_U32CTL, U32current);
6874         if (Writable == TRUE) {
6875             SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_DIMMED, FALSE);
6876             SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_VISIBLE, TRUE);
6877         }
6878         else if (Readable == TRUE) {
6879             SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_DIMMED, TRUE);
6880             SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_VISIBLE, TRUE);
6881         }
6882         else
6883             SetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_VISIBLE, FALSE);
6884     }
6885
6886     // For I64, get minimum, maximum, increment, and current value from driver, and configure the control
6887     // Note: Since CVI does not support 64-bit integer types, we will use doubles for it.
6888     else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeI64)
6889     {
6890         SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 1);
6891         GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 1, &TabPanelHandle);
6892         IMAQdxGetAttributeMinimum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLminimum);
6893         IMAQdxGetAttributeMaximum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLmaximum);
6894         IMAQdxGetAttributeIncrement (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLincrement);
6895         IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
6896         IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
6897         if (Readable == TRUE)
6898             IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLcurrent);
6899         SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_MIN_VALUE, DBLminimum);
6900         SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_MAX_VALUE, DBLmaximum);
6901         SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_INCR_VALUE, DBLincrement);
6902         SetCtrlVal (TabPanelHandle, I64TAB_I64CTL, DBLcurrent);
6903         if (Writable == TRUE) {
6904             SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_DIMMED, FALSE);
6905             SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_VISIBLE, TRUE);
6906         }
6907         else if (Readable == TRUE) {
6908             SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_DIMMED, TRUE);
6909             SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_VISIBLE, TRUE);
6910         }
6911         else
6912             SetCtrlAttribute (TabPanelHandle, I64TAB_I64CTL, ATTR_VISIBLE, FALSE);
6913     }
6914
6915     // For DBL (F64), get minimum, maximum and current value from driver, and configure the control
6916     else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeF64)
6917     {
6918         SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 2);
6919         GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 2, &TabPanelHandle);
6920         IMAQdxGetAttributeMinimum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLminimum);
6921         IMAQdxGetAttributeMaximum (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLmaximum);
6922         IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
6923         IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
6924         if (Readable == TRUE)
6925             IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, &DBLcurrent);
6926         SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_MIN_VALUE, DBLminimum);
6927         SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_MAX_VALUE, DBLmaximum);
6928         SetCtrlVal (TabPanelHandle, DBLTAB_DBLCTL, DBLcurrent);
6929         if (Writable == TRUE) {
6930             SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_DIMMED, FALSE);
6931             SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_VISIBLE, TRUE);
6932         }
6933         else if (Readable == TRUE) {
6934             SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_DIMMED, TRUE);
6935             SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_VISIBLE, TRUE);
```

```
6936            }
6937            else
6938                SetCtrlAttribute (TabPanelHandle, DBLTAB_DBLCTL, ATTR_VISIBLE, FALSE);
6939        }
6940
6941        // For String, get current value from driver, display current value in control
6942        else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeString)
6943        {
6944            SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 3);
6945            GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 3, &TabPanelHandle);
6946            IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
6947            IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
6948            if (Readable == TRUE)
6949                IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeString, STRcurrent);
6950            SetCtrlVal (TabPanelHandle, STRTAB_STRCTL, STRcurrent);
6951            if (Writable == TRUE) {
6952                SetCtrlAttribute (TabPanelHandle, STRTAB_STRCTL, ATTR_DIMMED, FALSE);
6953                SetCtrlAttribute (TabPanelHandle, STRTAB_STRCTL, ATTR_VISIBLE, TRUE);
6954            }
6955            else if (Readable == TRUE) {
6956                SetCtrlAttribute (TabPanelHandle, STRTAB_STRCTL, ATTR_DIMMED, TRUE);
6957                SetCtrlAttribute (TabPanelHandle, STRTAB_STRCTL, ATTR_VISIBLE, TRUE);
6958            }
6959            else
6960                SetCtrlAttribute (TabPanelHandle, STRTAB_STRCTL, ATTR_VISIBLE, FALSE);
6961        }
6962
6963        // For Enum, get the list of available items and the current value from driver, populate the control
6964        else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeEnum)
6965        {
6966            unsigned int EnumCount = 0;
6967            IMAQdxEnumItem* EnumList = NULL;
6968
6969            SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 4);
6970            GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 4, &TabPanelHandle);
6971            ClearListCtrl (TabPanelHandle, ENUMTAB_ENUMCTL);
6972            IMAQdxEnumerateAttributeValues (session, cameraAttributes[selectedAttrIndex].Name, NULL, &EnumCount);
6973            EnumList = (IMAQdxEnumItem *)malloc(EnumCount * sizeof(IMAQdxEnumItem));
6974            IMAQdxEnumerateAttributeValues (session, cameraAttributes[selectedAttrIndex].Name, EnumList, &EnumCount);
6975            for (i=0; i < EnumCount; i++)
6976            {
6977                InsertListItem (TabPanelHandle, ENUMTAB_ENUMCTL, -1, EnumList[i].Name, EnumList[i].Value);
6978            }
6979            IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
6980            IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
6981            if (Readable == TRUE)
6982                IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, &EnumCurrent);

6983            SetCtrlVal (TabPanelHandle, ENUMTAB_ENUMCTL, EnumCurrent);
6984            if (Writable == TRUE) {
6985                SetCtrlAttribute (TabPanelHandle, ENUMTAB_ENUMCTL, ATTR_DIMMED, FALSE);
6986                SetCtrlAttribute (TabPanelHandle, ENUMTAB_ENUMCTL, ATTR_VISIBLE, TRUE);
6987            }
6988            else if (Readable == TRUE) {
6989                SetCtrlAttribute (TabPanelHandle, ENUMTAB_ENUMCTL, ATTR_DIMMED, TRUE);
6990                SetCtrlAttribute (TabPanelHandle, ENUMTAB_ENUMCTL, ATTR_VISIBLE, TRUE);
6991            }
6992            else
6993                SetCtrlAttribute (TabPanelHandle, ENUMTAB_ENUMCTL, ATTR_VISIBLE, FALSE);
6994            if (EnumList)
6995            {
6996                free (EnumList);
6997                EnumList = NULL;
6998            }
6999        }
7000
7001        // For Boolean, get current value from driver, display current value in control
7002        else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeBool)
7003        {
7004            SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 5);
7005            GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 5, &TabPanelHandle);
7006            IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
7007            IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
7008            if (Readable == TRUE)
7009                IMAQdxGetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeBool, &BOOLcurrent);
7010            SetCtrlVal (TabPanelHandle, BOOLTAB_BOOLCTL, BOOLcurrent);
7011            if (Writable == TRUE) {
7012                SetCtrlAttribute (TabPanelHandle, BOOLTAB_BOOLCTL, ATTR_DIMMED, FALSE);
7013                SetCtrlAttribute (TabPanelHandle, BOOLTAB_BOOLCTL, ATTR_VISIBLE, TRUE);
7014            }
7015            else if (Readable == TRUE) {
7016                SetCtrlAttribute (TabPanelHandle, BOOLTAB_BOOLCTL, ATTR_DIMMED, TRUE);
7017                SetCtrlAttribute (TabPanelHandle, BOOLTAB_BOOLCTL, ATTR_VISIBLE, TRUE);
7018            }
7019            else
7020                SetCtrlAttribute (TabPanelHandle, BOOLTAB_BOOLCTL, ATTR_VISIBLE, FALSE);
7021        }
7022
7023        // For Command
7024        else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeCommand)
7025        {
7026            SetActiveTabPage (panelHandle, PANEL_CAM_TAB, 6);
7027            GetPanelHandleFromTabPage (panelHandle, PANEL_CAM_TAB, 6, &TabPanelHandle);
7028            IMAQdxIsAttributeReadable (session, cameraAttributes[selectedAttrIndex].Name, &Readable);
7029            IMAQdxIsAttributeWritable (session, cameraAttributes[selectedAttrIndex].Name, &Writable);
7030            if (Writable == TRUE) {
7031                SetCtrlAttribute (TabPanelHandle, CMDTAB CMDCTL, ATTR DIMMED, FALSE);
```

```
7032            SetCtrlAttribute (TabPanelHandle, CMDTAB_CMDCTL, ATTR_VISIBLE, TRUE);
7033        }
7034        else if (Readable == TRUE) {
7035            SetCtrlAttribute (TabPanelHandle, CMDTAB_CMDCTL, ATTR_DIMMED, TRUE);
7036            SetCtrlAttribute (TabPanelHandle, CMDTAB_CMDCTL, ATTR_VISIBLE, TRUE);
7037        }
7038        else
7039            SetCtrlAttribute (TabPanelHandle, CMDTAB_CMDCTL, ATTR_VISIBLE, FALSE);
7040    }
7041
7042    //SetCtrlVal(panelHandle, PANEL_LED_EXP_8, 1);
7043 }
7044
7045
7046 int CVICALLBACK Flush (int panel, int control, int event,
7047         void *callbackData, int eventData1, int eventData2)
7048 {
7049     unsigned int        U32current = 0;
7050     unsigned int        U32increment = 0;
7051     double              DBLcurrent = 0;
7052     double              DBLincrement = 0;
7053     char                STRcurrent[IMAQDX_MAX_API_STRING_LENGTH];
7054     unsigned int        EnumCurrent = 0;
7055     BOOL                BOOLcurrent = FALSE;
7056
7057     switch (event)
7058     {
7059         case EVENT_COMMIT:
7060
7061             // Get active attribute from table
7062             GetCtrlVal (panelHandle, PANEL_CAM_ATTR, &selectedAttrIndex);
7063
7064             //  Read the new value from the control, and write to the driver
7065             if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeU32)
7066             {
7067                 GetCtrlVal (TabPanelHandle, U32TAB_U32CTL, &U32current);
7068              // Silently coerce to the increment value and update the control
7069                 GetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_INCR_VALUE, &U32increment);
7070                 U32current = (unsigned int)(U32current / U32increment + 0.5) * U32increment;
7071                 SetCtrlVal (TabPanelHandle, U32TAB_U32CTL, U32current);
7072                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, U32current);
7073             }
7074             else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeI64)
7075             {
7076                 GetCtrlVal (TabPanelHandle, I64TAB_I64CTL, &DBLcurrent);
7077              // Silently coerce to the increment value and update the control
7078                 GetCtrlAttribute (TabPanelHandle, U32TAB_U32CTL, ATTR_INCR_VALUE, &DBLincrement);
7079                 DBLcurrent = (unsigned int)(DBLcurrent / DBLincrement + 0.5) * DBLincrement;
7080                 SetCtrlVal (TabPanelHandle, I64TAB_I64CTL, DBLcurrent);
7081                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, DBLcurrent);
7082             }
7083             else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeF64)
7084             {
7085                 GetCtrlVal (TabPanelHandle, DBLTAB_DBLCTL, &DBLcurrent);
7086                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeF64, DBLcurrent);
7087             }
7088             else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeString)
7089             {
7090                 GetCtrlVal (TabPanelHandle, STRTAB_STRCTL, &STRcurrent);
7091                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeString, STRcurrent);
7092             }
7093             else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeEnum)
7094             {
7095                 GetCtrlVal (TabPanelHandle, ENUMTAB_ENUMCTL, &EnumCurrent);
7096                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeU32, EnumCurrent);
7097             }
7098             else if (cameraAttributes[selectedAttrIndex].Type == IMAQdxAttributeTypeBool)
7099             {
7100                 GetCtrlVal (TabPanelHandle, BOOLTAB_BOOLCTL, &BOOLcurrent);
7101                 IMAQdxSetAttribute (session, cameraAttributes[selectedAttrIndex].Name, IMAQdxValueTypeBool, BOOLcurrent);
7102             }
7103             break;
7104     }
7105     return 0;
7106 }
7107
7108 int CVICALLBACK Snap (int panel, int control, int event,
7109         void *callbackData, int eventData1, int eventData2)
7110 {  int a;
7111     switch (event)
7112     {
7113         case EVENT_COMMIT:  // Creates an image
7114             SetCtrlAttribute (panel_handle, MAIN_Save, ATTR_DIMMED, 1);
7115             if (!image)
7116
7117             {
7118                 SetCtrlAttribute (panel_handle, MAIN_Save, ATTR_DIMMED, 1);
7119                 image = imaqCreateImage (IMAQ_IMAGE_U8, 0);
7120
7121             // snap a picture
7122             error2 = IMAQdxSnap (session, image);
7123
7124
7125             if (error2)
7126             {
7127                 DisplayError(error2);
7128                 break;
```

```
7129            }
7130
7131            // Display using IMAQ Vision
7132            imaqDisplayImage (image, 0, TRUE);
7133
7134            // Set settings to the NI Vision window
7135            if (Just_Grabbed == TRUE)
7136            {
7137                DSN_Init_Image(panelHandle);
7138                Just_Grabbed = 0;
7139            }
7140
7141            SetCtrlAttribute (panel_handle, MAIN_Save, ATTR_DIMMED, 0);
7142            break;
7143
7144            }
7145
7146        if(image)
7147            {
7148                SetCtrlAttribute (panel_handle, MAIN_Save, ATTR_DIMMED, 1);
7149               imaqDispose(image);
7150               image = NULL;
7151               image = imaqCreateImage (IMAQ_IMAGE_U8, 0);
7152
7153            // snap a picture
7154            error2 = IMAQdxSnap (session, image);
7155            if (error2)
7156            {
7157                break;
7158            }
7159
7160            // Display using IMAQ Vision
7161            imaqDisplayImage (image, 0, TRUE);
7162            SetCtrlAttribute (panel_handle, MAIN_Save, ATTR_DIMMED, 0);
7163            break;
7164            }
7165    }
7166    return 0;
7167 }
7168
7169 int CVICALLBACK save (int panel, int control, int event,
7170        void *callbackData, int eventData1, int eventData2)
7171 {
7172 char filename[MAX_PATHNAME_LEN];
7173    switch (event)
7174    {
7175        case EVENT_COMMIT:
7176             imaqWriteTIFFFile (image,"DATA\\Stage2_time3.tiff", FALSE, NULL);
7177        break;
7178    }
7179 return 0;
7180 }
7181
7182 int CVICALLBACK ReSetCallBack (int panel, int control, int event,
7183        void *callbackData, int eventData1, int eventData2)
7184 {
7185    switch (event)
7186    {
7187        case EVENT_COMMIT:
7188                DSN_Init_Image(panelHandle);
7189            break;
7190    }
7191    return 0;
7192 }
7193
7194 int CVICALLBACK Imaq_loop2 (int panel, int control, int event,
7195        void *callbackData, int eventData1, int eventData2)
7196 {
7197    unsigned int bufferNumber;
7198    char  setpoint[100], wholeNum_string[4], decNum_string[4];
7199    char* setpointcode = "DATA\\Stage";
7200    IMAQdxError error2 = IMAQdxErrorSuccess;
7201
7202    switch (event)
7203    {
7204        case EVENT_TIMER_TICK:
7205            // Get the next frame
7206            error2 = IMAQdxGrab (session, image_g, TRUE, &bufferNumber);
7207            if (error2)
7208            {
7209                break;
7210            }
7211
7212            // Display using NI Vision
7213            imaqDisplayImage (image_g, 0, TRUE);
7214
7215    //      wholeNum=wholeNum+1;
7216    //      decNum=decNum+1;
7217
7218            sprintf(wholeNum_string, "%i", wholeNum);
7219            sprintf(decNum_string, "%i", decNum);
7220
7221            sprintf(setpoint, "DATA\\Stage%dTtube%2.0fWIn%2.3fOIn%2.3fWInR%2.3fEInR%2.3f.tiff",
7222                   stage, MANUAL_TEST_TUBE[0], WATERINJPV[1], OILINJPV[1] ,WATERINJ2PV[1], EMULINJPV[1]);
7223            //Description for acronyms:
7224            //Ttube - test tube number
7225            //WIn   - water flooding during initial saturation process and permeability measurements
```

```
7226            //OIn   - oil injected during saturation process
7227            //WInR  - water flooding during recovery process
7228            //EInR  - emulsion flooding during recovery process
7229
7230
7231            imaqWriteTIFFFile (image_g, setpoint, FALSE, NULL);
7232
7233            // Set settings to the NI Vision window
7234            if (Just_Grabbed == TRUE)
7235            {
7236                DSN_Init_Image(panelHandle);
7237                Just_Grabbed = 0;
7238            }
7239            break;
7240    }
7241 if (error2) { DisplayError(error2); }
7242 return 0;
7243 }
7244
7245 int CVICALLBACK SnapSave (int panel, int control, int event,
7246         void *callbackData, int eventData1, int eventData2)
7247 {
7248 int test,i;
7249 double time;
7250     switch (event)
7251     {
7252         case EVENT_COMMIT:
7253             IMAQdxUnconfigureAcquisition(session);
7254             IMAQdxConfigureGrab (session);
7255             GetCtrlVal (panel_handle, MAIN_SNAP_SAVE, &test);
7256
7257             if(test)
7258             {
7259                 // Create an image
7260                 if(image_g)
7261                     imaqDispose(image_g);
7262                 image_g = imaqCreateImage (IMAQ_IMAGE_U8, 0);
7263                 // Configure the acquisiton
7264                 IMAQdxConfigureAcquisition (session, TRUE, 3);
7265                 // Start the acquisition
7266                 IMAQdxStartAcquisition (session);
7267                 // RUN the timer to do the GRAB and DISPLAY
7268             //  SetCtrlAttribute (panel_handle, MAIN_TIMER_2, ATTR_ENABLED, TRUE);
7269                 Is_Cam= 1;
7270                 SetCtrlVal(panel_handle,MAIN_LOG_CAM, Is_Cam);
7271                 SetCtrlAttribute(panel_handle,MAIN_SNAP,  ATTR_DIMMED,      1);
7272                 SetCtrlAttribute(panel_handle,MAIN_Save,  ATTR_DIMMED,      1);
7273                 SetCtrlAttribute(panel_handle,MAIN_GRAB_2,ATTR_DIMMED,      1);
7274             }
7275             else
7276             {
7277                 // Stop the Imaq Loop
7278             //  SetCtrlAttribute (panel_handle, MAIN_TIMER_2, ATTR_ENABLED, FALSE);
7279                 // Stop the acquisition
7280                 IMAQdxStopAcquisition (session);
7281                 // Unconfigure the acquisition
7282                 IMAQdxUnconfigureAcquisition (session);
7283
7284                 Is_Cam= 0;
7285                 SetCtrlVal(panel_handle,MAIN_LOG_CAM, Is_Cam);
7286                 SetCtrlAttribute(panel_handle,MAIN_SNAP,  ATTR_DIMMED,      0);
7287                 SetCtrlAttribute(panel_handle,MAIN_Save,  ATTR_DIMMED,      0);
7288                 SetCtrlAttribute(panel_handle,MAIN_GRAB_2,ATTR_DIMMED,      0);
7289
7290                 if(image_g)
7291                 {
7292                     imaqDispose(image_g);
7293                     image_g = NULL;
7294                 }
7295             }
7296             break;
7297     }
7298     return 0;
7299 }
7300
7301 int CVICALLBACK SetTimer (int panel, int control, int event,
7302         void *callbackData, int eventData1, int eventData2)
7303 {
7304     double temp,temp2;
7305     switch (event)
7306     {
7307         case EVENT_COMMIT:
7308             GetCtrlVal(panel_handle,MAIN_NUMERIC,&temp);
7309             temp2 = temp * 60;
7310             SetCtrlAttribute (panel_handle, MAIN_TIMER_2, ATTR_INTERVAL, temp2);
7311             break;
7312     }
7313     return 0;
7314 }
7315
7316 int CVICALLBACK OnAttributeSelect (int panel, int control, int event,
7317         void *callbackData, int eventData1, int eventData2)
7318 {
7319     switch (event)
7320     {
7321         case EVENT_VAL_CHANGED:
7322             SetCtrlVal(panelHandle, PANEL LED EXP 6, 0);
```

```
7323             SetCtrlVal(panelHandle, PANEL_LED_EXP_7, 0);
7324             SetCtrlVal(panelHandle, PANEL_LED_EXP_8, 0);
7325             SetCtrlVal(panelHandle, PANEL_LED_EXP_9, 0);
7326
7327             UpdateAttributeTab();
7328             break;
7329     }
7330     return 0;
7331 }
7332
7333 int CVICALLBACK QUICK_CAM_ATT (int panel, int control, int event,
7334         void *callbackData, int eventData1, int eventData2)
7335 {
7336     int line_num =0;
7337     switch (event)
7338     {
7339         case EVENT_COMMIT:
7340         switch (control)
7341         {
7342                 case  PANEL_QUICK_EXP:
7343                       SetCtrlVal (panelHandle, PANEL_CAM_ATTR, 35);
7344                       break;
7345                 case  PANEL_QUICK_GAIN:
7346                       SetCtrlVal (panelHandle, PANEL_CAM_ATTR, 41);
7347                       break;
7348                 case  PANEL_QUICK_OTHER:
7349                       GetCtrlVal (panelHandle, PANEL_Other_Num, &line_num);
7350                       SetCtrlVal (panelHandle, PANEL_CAM_ATTR, line_num);
7351                       break;
7352                 case PANEL_QUICK_AOI:
7353                       SetCtrlVal (panelHandle, PANEL_CAM_ATTR, 272);
7354                       break;
7355                 case PANEL_QUICK_BPP:
7356                       SetCtrlVal (panelHandle, PANEL_CAM_ATTR, 23);
7357                       break;
7358         }
7359         UpdateAttributeTab();
7360         break;
7361     }
7362 return 0;
7363 }
7364
7365 int CVICALLBACK ClosePreCallBack (int panel, int control, int event,
7366         void *callbackData, int eventData1, int eventData2)
7367 {
7368     switch (event)
7369     {
7370         case EVENT_COMMIT:
7371                         HidePanel (pressure_handle);
7372             break;
7373     }
7374     return 0;
7375 }
7376
7377 int CVICALLBACK PressureMapCallBack (int panel, int control, int event,
7378         void *callbackData, int eventData1, int eventData2)
7379 {
7380     switch (event)
7381     {
7382         case EVENT_COMMIT:
7383                       DisplayPanel(pressure_handle);
7384             break;
7385     }
7386     return 0;
7387 }
7388
7389 int CVICALLBACK OpenServiceCallBack (int panel, int control, int event,
7390         void *callbackData, int eventData1, int eventData2)
7391 {
7392     switch (event)
7393     {
7394         case EVENT_COMMIT:
7395                                       DisplayPanel(pumpservice_handle);
7396             break;
7397     }
7398     return 0;
7399 }
7400
7401 int CVICALLBACK ClosePumpSeviceCallback (int panel, int control, int event,
7402         void *callbackData, int eventData1, int eventData2)
7403 {
7404     switch (event)
7405     {
7406         case EVENT_COMMIT:
7407                         HidePanel (pumpservice_handle);
7408             break;
7409     }
7410     return 0;
7411 }
7412
7413 int CVICALLBACK CheckMinPCallBack (int panel, int control, int event,
7414         void *callbackData, int eventData1, int eventData2)
7415 {
7416 int i,pump;
7417 double probe_temp;
7418 char temp [NUM];
7419
```

```
7420 switch (event)
7421 {
7422         case EVENT_COMMIT:
7423
7424
7425                             ////////////
7426                             panel_name=pumpservice_handle;
7427                             GetCtrlVal(pumpservice_handle,PUMP_SER_COMNUMBER, &pump);
7428                             if (pump == 1)      CP=P1_comport;
7429                             else if (pump == 2) CP=P2_comport;
7430                             else if (pump == 3) CP=P3_comport;
7431                             else if (pump == 0)
7432                             {
7433                             MessagePopup ("Warning","Please select the pump.");
7434                             break;
7435                             }
7436
7437                             text_panel_name= PUMP_SER_PRESSURE_MIN_READ;
7438                             ////////////
7439
7440                             FlushOutQ (CP);
7441                             FlushInQ (CP);
7442
7443                             send_data[0] = '\0';
7444                             Fmt (send_data, "%s<PMIN10?\r");
7445                             stringsize = StringLength (send_data);
7446
7447                             bytes_sent = ComWrt (CP, send_data, stringsize);
7448                             Delay(0.05); /* 0.05  */
7449
7450                             read_cnt = GetInQLen (CP);
7451
7452                             read_data[0] = '\0';
7453                             read_term = 13;
7454
7455                             bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
7456                             CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
7457                         /*
7458                                 for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
7459                                     temp[i-2]=tbox_read_data[i];
7460                                     temp[i-2]='\0';         //close by '\0'
7461
7462                                 probe_temp = atof(temp);
7463                             ProcessSystemEvents ();
7464                                 SetCtrlVal (panel_name, text_panel_name, probe_temp);     */
7465
7466                         SetCtrlVal (panel_name, text_panel_name, tbox_read_data); //temporary
7467
7468                             RS232Error = ReturnRS232Err ();
7469                             if (RS232Error)
7470                                 DisplayRS232Error ();
7471
7472                             send_data[0] = '\0';
7473             break;
7474 }
7475     return 0;
7476 }
7477
7478 int CVICALLBACK CheckMaxPCallBack (int panel, int control, int event,
7479         void *callbackData, int eventData1, int eventData2)
7480 {
7481 int i,pump;
7482 double probe_temp;
7483 char temp [NUM];
7484
7485 switch (event)
7486 {
7487         case EVENT_COMMIT:
7488                             ////////////
7489                             panel_name=pumpservice_handle;
7490                             GetCtrlVal(pumpservice_handle,PUMP_SER_COMNUMBER, &pump);
7491                             if (pump == 1)      CP=P1_comport;
7492                             else if (pump == 2) CP=P2_comport;
7493                             else if (pump == 3) CP=P3_comport;
7494                             else if (pump == 0)
7495                             {
7496                             MessagePopup ("Warning","Please select the pump.");
7497                             break;
7498                             }
7499
7500                             text_panel_name= PUMP_SER_PRESSURE_MIN_READ;
7501                             ////////////
7502
7503                             FlushOutQ (CP);
7504                             FlushInQ (CP);
7505
7506                             send_data[0] = '\0';
7507                             Fmt (send_data, "%s<PMAX10?\r");
7508                             stringsize = StringLength (send_data);
7509
7510                             bytes_sent = ComWrt (CP, send_data, stringsize);
7511                             Delay(0.05); /* 0.05  */
7512
7513                             read_cnt = GetInQLen (CP);
7514
7515                             read_data[0] = '\0';
7516                             read_term = 13;
```

```
7517
7518                            bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
7519                            CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
7520                        /*
7521                                for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
7522                                    temp[i-2]=tbox_read_data[i];
7523                                    temp[i-2]='\0';         //close by '\0'
7524
7525                                probe_temp = atof(temp);
7526                            ProcessSystemEvents ();
7527                                SetCtrlVal (panel_name, text_panel_name, probe_temp);     */
7528
7529                        SetCtrlVal (panel_name, text_panel_name, tbox_read_data); //temporary
7530
7531                            RS232Error = ReturnRS232Err ();
7532                            if (RS232Error)
7533                                DisplayRS232Error ();
7534
7535                            send_data[0] = '\0';
7536            break;
7537 }
7538     return 0;
7539 }
7540
7541
7542
7543 int CVICALLBACK SetMinPCallBack (int panel, int control, int event,
7544         void *callbackData, int eventData1, int eventData2)
7545 { int pump;
7546     switch (event)
7547     {
7548         case EVENT_COMMIT:
7549                 GetCtrlVal(pumpservice_handle,PUMP_SER_COMNUMBER, &pump);
7550                 if (pump == 1)      CP=P1_comport;
7551                 else if (pump == 2) CP=P2_comport;
7552                 else if (pump == 3) CP=P3_comport;
7553                 else if (pump == 0)
7554                         {
7555                         MessagePopup ("Warning","Please select the pump.");
7556                         break;
7557                         }
7558
7559             GetCtrlVal(pumpservice_handle, PUMP_SER_PRESSURE_MIN_SEND, &pmin_setpoint);
7560             SetMinPressure();
7561             break;
7562     }
7563     return 0;
7564 }
7565
7566 int CVICALLBACK SetMaxPCallBack (int panel, int control, int event,
7567         void *callbackData, int eventData1, int eventData2)
7568 {int pump;
7569     switch (event)
7570     {
7571         case EVENT_COMMIT:
7572                 GetCtrlVal(pumpservice_handle,PUMP_SER_COMNUMBER, &pump);
7573                 if (pump == 1)      CP=P1_comport;
7574                 else if (pump == 2) CP=P2_comport;
7575                 else if (pump == 3) CP=P3_comport;
7576                 else if (pump == 0)
7577                         {
7578                         MessagePopup ("Warning","Please select the pump.");
7579                         break;
7580                         }
7581
7582             GetCtrlVal(pumpservice_handle, PUMP_SER_PRESSURE_MAX_SEND, &pmax_setpoint);
7583             SetMaxPressure();
7584         break;
7585     }
7586     return 0;
7587 }
7588
7589
7590
7591
7592
7593 void SetMaxPressure(void) /* build based on DSN_Set_Bath_Setpoint*/
7594 {
7595         //CP= current comport
7596         //LD= LED to set ON
7597
7598         char  ascii_setpoint[15],
7599               wholeNum_string[4],
7600               decNum_string[4];
7601
7602         char* setpointcode = "PMAX10:";
7603
7604
7605         int wholeNum,
7606             decNum;
7607
7608         wholeNum = pmax_setpoint;                       // Captures the whole number portion of the setpoint
7609         decNum = (pmax_setpoint-wholeNum)*1000;     // Captures the 2 decimal places of the setpoint
7610
7611         sprintf(wholeNum_string, "%i", wholeNum);
7612         sprintf(decNum_string, "%i", decNum);
7613
```

```
7614 if (decNum == 0)
7615         {
7616         // Assembles the command as shown below     e.g.
7617         strcpy(ascii_setpoint, setpointcode);       // P
7618         strcat(ascii_setpoint, wholeNum_string);    // P10
7619         strcat(ascii_setpoint, ".");                // P10.
7620         strcat(ascii_setpoint, decNum_string);      // P10.0
7621         strcat(ascii_setpoint, decNum_string);      // P10.00
7622         strcat(ascii_setpoint, decNum_string);      // P10.000
7623         strcat(ascii_setpoint, "\r");               // P10.000\r
7624         }
7625         else
7626         {
7627             // Assembles the command as shown below
7628             strcpy(ascii_setpoint, setpointcode);       // P
7629             strcat(ascii_setpoint, wholeNum_string);    // P10
7630             strcat(ascii_setpoint, ".");                // P10.
7631             strcat(ascii_setpoint, decNum_string);      // P10.XXX
7632             strcat(ascii_setpoint, "\r");               // P10.XXX\r
7633         }
7634
7635         SetCtrlVal(pumpservice_handle,PUMP_SER_PRESSURE_MAX_READ_2,ascii_setpoint);
7636
7637         FlushOutQ (CP);
7638         FlushInQ (CP);
7639
7640         stringsize = StringLength (ascii_setpoint);
7641         ComWrt (CP, ascii_setpoint, stringsize);    // Sends new setpoint to COM Port
7642
7643         Delay(0.5);
7644
7645         read_cnt = GetInQLen (CP);
7646
7647         read_term = 13;
7648
7649         bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
7650         CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
7651
7652
7653
7654     //  SetCtrlVal (panel_handle, SERIAL_FLOWRATE_R, tbox_read_data);
7655       /*
7656
7657         if (tbox_read_data && "OK")
7658                SetCtrlVal (panel_handle, LED, 1);
7659         else
7660         {
7661                SetCtrlVal (panel_handle, LED, 0);
7662                MessagePopup ("Error","flow rate has not changed");
7663         }
7664
7665         RS232Error = ReturnRS232Err ();
7666             if (RS232Error)
7667                 DisplayRS232Error ();
7668         // Return LED back to the initial value
7669         Delay(0.5);
7670         SetCtrlVal (panel_handle, LED, 0);   */
7671 }
7672
7673
7674
7675 void SetMinPressure(void) /* build based on DSN_Set_Bath_Setpoint*/
7676 {
7677         //CP= current comport
7678         //LD= LED to set ON
7679
7680         char  ascii_setpoint[15],
7681               wholeNum_string[4],
7682               decNum_string[4];
7683
7684         char* setpointcode = "PMIN10:";
7685
7686
7687         int wholeNum,
7688             decNum;
7689
7690         wholeNum = pmax_setpoint;                       // Captures the whole number portion of the setpoint
7691         decNum = (pmax_setpoint-wholeNum)*1000;     // Captures the 2 decimal places of the setpoint
7692
7693         sprintf(wholeNum_string, "%i", wholeNum);
7694         sprintf(decNum_string, "%i", decNum);
7695
7696 if (decNum == 0)
7697         {
7698         // Assembles the command as shown below     e.g.
7699         strcpy(ascii_setpoint, setpointcode);       // P
7700         strcat(ascii_setpoint, wholeNum_string);    // P10
7701         strcat(ascii_setpoint, ".");                // P10.
7702         strcat(ascii_setpoint, decNum_string);      // P10.0
7703         strcat(ascii_setpoint, decNum_string);      // P10.00
7704         strcat(ascii_setpoint, decNum_string);      // P10.000
7705         strcat(ascii_setpoint, "\r");               // P10.000\r
7706         }
7707         else
7708         {
7709             // Assembles the command as shown below
7710             strcpy(ascii setpoint, setpointcode);       // P
```

```
7711             strcat(ascii_setpoint, wholeNum_string);    // P10
7712             strcat(ascii_setpoint, ".");                // P10.
7713             strcat(ascii_setpoint, decNum_string);      // P10.XXX
7714             strcat(ascii_setpoint, "\r");               // P10.XXX\r
7715         }
7716
7717         SetCtrlVal(pumpservice_handle,PUMP_SER_PRESSURE_MAX_READ_2,ascii_setpoint);
7718
7719         FlushOutQ (CP);
7720         FlushInQ (CP);
7721
7722         stringsize = StringLength (ascii_setpoint);
7723         ComWrt (CP, ascii_setpoint, stringsize);    // Sends new setpoint to COM Port
7724
7725         Delay(0.5);
7726
7727         read_cnt = GetInQLen (CP);
7728
7729         read_term = 13;
7730
7731         bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
7732         CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
7733
7734
7735
7736     //  SetCtrlVal (panel_handle, SERIAL_FLOWRATE_R, tbox_read_data);
7737       /*
7738
7739         if (tbox_read_data && "OK")
7740                SetCtrlVal (panel_handle, LED, 1);
7741         else
7742         {
7743                SetCtrlVal (panel_handle, LED, 0);
7744                MessagePopup ("Error","flow rate has not changed");
7745         }
7746
7747         RS232Error = ReturnRS232Err ();
7748             if (RS232Error)
7749                  DisplayRS232Error ();
7750         // Return LED back to the initial value
7751         Delay(0.5);
7752         SetCtrlVal (panel_handle, LED, 0);   */
7753 }
```

```
////// Quadrise MCP_pump.c includes controls for Pump#2 and Pump#3
////// with same logic as written for Pump#1 in Quadrise MCP.c

#include <rs232.h>
#include <utility.h>
#include <formatio.h>
#include <userint.h>
#include "Quadrise MCP.h"
#include "Quadrise MCP_Declare.h"

//Extra stuff
void EnableP2PanelControls (int enable)   /* PUMP #1 buttons activation */
{
    //pump
    SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_START, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_STOP, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_PURGE, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_RESTART, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_5ERR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_SET_NFR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_SNFR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_STATUS_P_2, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P2_TEXTMSG, ATTR_DIMMED, enable);
}

void EnableP3PanelControls (int enable)   /* PUMP #3 buttons activation */
{
    //pump
    SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_START, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_STOP, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_PURGE, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_RESTART, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_5ERR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_SET_NFR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_SNFR, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_STATUS_P_3, ATTR_DIMMED, enable);
    SetCtrlAttribute (panel_handle, MAIN_P3_TEXTMSG, ATTR_DIMMED, enable);
}

////////////////////////////////
// Pump #3 On.Off/Start/Stop/Reset/5errors/Set new flow rate

int CVICALLBACK P3SNFRCallBack (int panel, int control, int event, //New Flow rate
        void *callbackData, int eventData1, int eventData2)
{

int i;
double probe_temp;
char temp [NUM];

    switch (event)
    {
        case EVENT_COMMIT:
CmtGetLock (Pumps_lockHandle);

               GetCtrlVal(panel_handle, MAIN_P3_SET_NFR, &wb_setpoint);
               CP=P3_comport;
               LED=MAIN_P3_OK_LED;

               SetPumpFlowrate();

               Delay(0.2);
               //update panel with new flow rate value//

                    ////////////
                    panel_name=panel_handle;
                    CP=P3_comport;
                    text_panel_name= MAIN_WAS_FR_P_3;
                    ////////////

                    FlushOutQ (CP);
                    FlushInQ (CP);

                    send_data[0] = '\0';
                    Fmt (send_data, "%s<F?\r");
                    stringsize = StringLength (send_data);

                    bytes_sent = ComWrt (CP, send_data, stringsize);
                    Delay(0.05); /* 0.05  */

                    read_cnt = GetInQLen (CP);

                    read_data[0] = '\0';
                    read_term = 13;

                    bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
                    CopyString (tbox_read_data, 0, read_data, 0, bytes_read);

                        for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
                            temp[i-2]=tbox_read_data[i];
                            temp[i-2]='\0';         //close by '\0'

                        probe_temp = atof(temp);
             ProcessSystemEvents ();
                        SetCtrlVal (panel name, text panel name, probe temp);
```

```
 98
 99                     PFR31[0] = probe_temp;      // Log value in PFR11 Array
100
101                     RS232Error = ReturnRS232Err ();
102                     if (RS232Error)
103                         DisplayRS232Error ();
104
105                     send_data[0] = '\0';
106
107 CmtReleaseLock (Pumps_lockHandle);
108             break;
109     }
110     return 0;
111 }
112
113 int CVICALLBACK P3ActivateCallBack (int panel, int control, int event,
114         void *callbackData, int eventData1, int eventData2)
115 {
116     switch (event)
117     {
118         case EVENT_COMMIT:
119             GetCtrlVal (panel_handle, MAIN_P3_ONOFF, &test);
120 if (p3check == 1)
121 {
122             if (Pumps_quitflag == 1)
123             {
124              CmtGetLock     (Pumps_lockHandle);
125                 ActivatePump3 ();
126              CmtReleaseLock (Pumps_lockHandle);
127             }
128             else ActivatePump3 ();
129 }
130         else
131         {
132         MessagePopup ("Warning","Connection to Pump #3 is required");
133         Delay(0.05);
134         SetCtrlVal(panel_handle, MAIN_P3_ONOFF, 0);
135         }
136             break;
137     }
138     return 0;
139 }
140
141 void ActivatePump3 (void)
142 {
143 int i;
144 double probe_temp;
145 char temp [NUM];
146             if(test==1)
147             {
148                     EnableP3PanelControls (0);
149                     SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_3, ATTR_VISIBLE, 1);
150                     SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_3, ATTR_VISIBLE, 1);
151                     SetCtrlVal (panel_handle, MAIN_P3_ONOFF_LED, 1);
152
153                     ////////////
154                     panel_name=panel_handle;
155                     CP=P3_comport;
156                     text_panel_name= MAIN_WAS_FR_P_3;
157                     ////////////
158
159                     FlushOutQ (CP);
160                     FlushInQ (CP);
161
162                     send_data[0] = '\0';
163                     Fmt (send_data, "%s<F?\r");
164                     stringsize = StringLength (send_data);
165
166                     bytes_sent = ComWrt (CP, send_data, stringsize);
167                     Delay(0.05); /* 0.05  */
168
169                     read_cnt = GetInQLen (CP);
170
171                     read_data[0] = '\0';
172                     read_term = 13;
173
174                     bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
175                     CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
176
177                         for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
178                             temp[i-2]=tbox_read_data[i];
179                             temp[i-2]='\0';         //close by '\0'
180
181                     probe_temp = atof(temp);
182                     ProcessSystemEvents ();
183                     SetCtrlVal (panel_name, text_panel_name, probe_temp);
184
185                     PFR31[0] = probe_temp;      // Log value in PFR31 Array
186
187                     RS232Error = ReturnRS232Err ();
188                     if (RS232Error)
189                         DisplayRS232Error ();
190
191                     send_data[0] = '\0';
192
193                     SetCtrlVal(panel_handle, MAIN_LOG_P3, 1);   //activate logging pressure
194                     GetCtrlVal(panel_handle, MAIN_LOG_P3, &Is_P3);
```

```
            }
            else
            {
                SetCtrlVal(panel_handle, MAIN_LOG_P3, 0);       //deactivate logging pressure
                GetCtrlVal(panel_handle, MAIN_LOG_P3, &Is_P3);  //save new value

                EnableP3PanelControls (1);  //dim all panels
                SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_3, ATTR_VISIBLE, 0);
                SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_3, ATTR_VISIBLE, 0);
                SetCtrlVal (panel_handle, MAIN_P3_ONOFF_LED, 0);

                    ////////////
                    panel_name=panel_handle;
                    CP=P3_comport;
                    PTEXT_panel_name= MAIN_STATUS_P_3;
                    ////////////

                send_data[0] = '\0';
                Fmt (send_data, "%s<OFF\r");
                SendReadPump ();
                PFR31[0] = '\0';        // Log value in PFR11 Array         }
            }
return;
}



int CVICALLBACK P3SFCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
CmtGetLock (Pumps_lockHandle);
                Delay (0.1);
                panel_name=panel_handle;
                CP=P3_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<ON\r");
                PTEXT_panel_name= MAIN_STATUS_P_3; //show response from pump
                SendReadPump ();

                pumprun_3 = 31;

CmtReleaseLock (Pumps_lockHandle);
            break;
        case EVENT_RIGHT_CLICK :
            MessagePopup ("Pump #3", "Will run Pump at current flow rate.");
            break;
    }
    return 0;
}

int CVICALLBACK P3STOPCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
CmtGetLock (Pumps_lockHandle);
                Delay (0.1);
                panel_name=panel_handle;
                CP=P3_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<OFF\r");
                PTEXT_panel_name= MAIN_STATUS_P_3;
                SendReadPump ();

                pumprun_3 = 32;

CmtReleaseLock (Pumps_lockHandle);
            break;
    }
    return 0;
}

int CVICALLBACK P3PURGECallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
                panel_name=panel_handle;
                CP=P3_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<PURGE\r");
                PTEXT_panel_name= MAIN_STATUS_P_3;
                SendReadPump ();
            break;
    }
    return 0;
}

int CVICALLBACK P3RESTARTCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
```

```
292     {
293         case EVENT_COMMIT:
294                panel_name=panel_handle;
295                CP=P3_comport;
296                send_data[0] = '\0';
297                Fmt (send_data, "%s<RESET\r");
298                PTEXT_panel_name= MAIN_STATUS_P_3;
299                SendReadPump ();
300            break;
301     }
302     return 0;
303 }
304
305 int CVICALLBACK P35ERRCallBack (int panel, int control, int event,
306         void *callbackData, int eventData1, int eventData2)
307 {
308     switch (event)
309     {
310         case EVENT_COMMIT:
311                panel_name=panel_handle;
312                CP=P3_comport;
313                send_data[0] = '\0';
314                Fmt (send_data, "%s<ERRORS?\r");
315                PTEXT_panel_name= MAIN_STATUS_P_3;
316                SendReadPump ();
317            break;
318     }
319     return 0;
320 }
321
322
323 ////////////////////////////////////////////////////////////
324 // Pump #3 On.Off/Start/Stop/Reset/5errors/Set new flow rate
325 ////////////////////////////////////////////////////////////
326
327
328 ////////////////////////////////////////////////////////////
329 // Pump #2 On.Off/Start/Stop/Reset/5errors/Set new flow rate
330 ////////////////////////////////////////////////////////////
331
332
333 int CVICALLBACK P2SNFRCallBack (int panel, int control, int event,
334         void *callbackData, int eventData1, int eventData2)
335 {
336
337 int i;
338 double probe_temp;
339 char temp [NUM];
340
341     switch (event)
342     {
343         case EVENT_COMMIT:
344 CmtGetLock (Pumps_lockHandle);
345
346                GetCtrlVal(panel_handle, MAIN_P2_SET_NFR, &wb_setpoint);
347                CP=P2_comport;
348                LED=MAIN_P2_OK_LED;
349
350                SetPumpFlowrate();
351
352                Delay(0.2);
353                //update panel with new flow rate value//
354
355                     ////////////
356                     panel_name=panel_handle;
357                     CP=P2_comport;
358                     text_panel_name= MAIN_WAS_FR_P_2;
359                     ////////////
360
361                     FlushOutQ (CP);
362                     FlushInQ (CP);
363
364                     send_data[0] = '\0';
365                     Fmt (send_data, "%s<F?\r");
366                     stringsize = StringLength (send_data);
367
368                     bytes_sent = ComWrt (CP, send_data, stringsize);
369                     Delay(0.05); /* 0.05  */
370
371                     read_cnt = GetInQLen (CP);
372
373                     read_data[0] = '\0';
374                     read_term = 13;
375
376                     bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
377                     CopyString (tbox_read_data, 0, read_data, 0, bytes_read);
378
379                         for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
380                             temp[i-2]=tbox_read_data[i];
381                             temp[i-2]='\0';         //close by '\0'
382
383                         probe_temp = atof(temp);
384             ProcessSystemEvents ();
385                         SetCtrlVal (panel_name, text_panel_name, probe_temp);
386
387                     PFR21[0] = probe_temp;      // Log value in PFR11 Array
388
```

```
                    RS232Error = ReturnRS232Err ();
                    if (RS232Error)
                        DisplayRS232Error ();

                    send_data[0] = '\0';


CmtReleaseLock (Pumps_lockHandle);
            break;
    }
    return 0;
}

int CVICALLBACK P2ActivateCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
switch (event)
    {
    case EVENT_COMMIT:
    GetCtrlVal (panel_handle, MAIN_P2_ONOFF, &test);
        if (p2check == 1)
        {
                if (Pumps_quitflag == 1)
                {
                 CmtGetLock     (Pumps_lockHandle);
                    ActivatePump2 ();
                 CmtReleaseLock (Pumps_lockHandle);
                }
                else ActivatePump2 ();
        }
            else
            {
            MessagePopup ("Warning","Connection to Pump #2 is required");
            Delay(0.05);
            SetCtrlVal(panel_handle, MAIN_P2_ONOFF, 0);
            }
    break;
    }
return 0;
}

void ActivatePump2 (void)
{
int i;
double probe_temp;
char temp [NUM];
            if(test==1)
            {

                    EnableP2PanelControls (0);
                    SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_2, ATTR_VISIBLE, 1);
                    SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_2, ATTR_VISIBLE, 1);
                    SetCtrlVal (panel_handle, MAIN_P2_ONOFF_LED, 1);

                    ////////////
                    panel_name=panel_handle;
                    CP=P2_comport;
                    text_panel_name= MAIN_WAS_FR_P_2;
                    ///////////

                    FlushOutQ (CP);
                    FlushInQ (CP);

                    send_data[0] = '\0';
                    Fmt (send_data, "%s<F?\r");
                    stringsize = StringLength (send_data);

                    bytes_sent = ComWrt (CP, send_data, stringsize);
                    Delay(0.05); /* 0.05  */

                    read_cnt = GetInQLen (CP);

                    read_data[0] = '\0';
                    read_term = 13;

                    bytes_read = ComRdTerm (CP, read_data, read_cnt, read_term);
                    CopyString (tbox_read_data, 0, read_data, 0, bytes_read);

                        for (i=2; tbox_read_data[i]!='\0';i++)  //Return is "Pressure:XX"
                            temp[i-2]=tbox_read_data[i];
                            temp[i-2]='\0';         //close by '\0'

                        probe_temp = atof(temp);
                    ProcessSystemEvents ();
                        SetCtrlVal (panel_name, text_panel_name, probe_temp);

                    PFR21[0] = probe_temp;      // Log value in PFR21 Array

                    RS232Error = ReturnRS232Err ();
                    if (RS232Error)
                        DisplayRS232Error ();

                    send_data[0] = '\0';

                    SetCtrlVal(panel_handle, MAIN_LOG_P2, 1);   //activate logging pressure
                    GetCtrlVal(panel_handle, MAIN_LOG_P2, &Is_P2);
            }
```

```
            else
            {
                SetCtrlVal(panel_handle, MAIN_LOG_P2, 0);       //deactivate logging pressure
                GetCtrlVal(panel_handle, MAIN_LOG_P2, &Is_P2);  //save new value

                EnableP2PanelControls (1);  //dim all panels
                SetCtrlAttribute (panel_handle, MAIN_WAS_FR_P_2, ATTR_VISIBLE, 0);
                SetCtrlAttribute (panel_handle, MAIN_PRESSURE_P_2, ATTR_VISIBLE, 0);
                SetCtrlVal (panel_handle, MAIN_P2_ONOFF_LED, 0);

                    ////////////
                    panel_name=panel_handle;
                    CP=P2_comport;
                    PTEXT_panel_name= MAIN_STATUS_P_2;
                    ////////////

                send_data[0] = '\0';
                Fmt (send_data, "%s<OFF\r");
                SendReadPump ();
                PFR21[0] = '\0';        // Log value in PFR11 Array
            }
return;
}


int CVICALLBACK P2SFCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
CmtGetLock (Pumps_lockHandle);
                Delay (0.1);
                panel_name=panel_handle;
                CP=P2_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<ON\r");
                PTEXT_panel_name= MAIN_STATUS_P_2; //show response from pump
                SendReadPump ();

                pumprun_2 = 21;

CmtReleaseLock (Pumps_lockHandle);
            break;
        case EVENT_RIGHT_CLICK :
            MessagePopup ("Pump #2", "Will run Pump at current flow rate.");
            break;
    }
    return 0;
}

int CVICALLBACK P2STOPCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
CmtGetLock (Pumps_lockHandle);
                Delay (0.1);
                panel_name=panel_handle;
                CP=P2_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<OFF\r");
                PTEXT_panel_name= MAIN_STATUS_P_2;
                SendReadPump ();

                pumprun_2 = 22;

CmtReleaseLock (Pumps_lockHandle);
            break;
    }
    return 0;
}

int CVICALLBACK P2PURGECallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
                panel_name=panel_handle;
                CP=P2_comport;
                send_data[0] = '\0';
                Fmt (send_data, "%s<PURGE\r");
                PTEXT_panel_name= MAIN_STATUS_P_2;
                SendReadPump ();
            break;
    }
    return 0;
}

int CVICALLBACK P2RESTARTCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT COMMIT:
```

```
583                 panel_name=panel_handle;
584                 CP=P2_comport;
585                 send_data[0] = '\0';
586                 Fmt (send_data, "%s<RESET\r");
587                 PTEXT_panel_name= MAIN_STATUS_P_2;
588                 SendReadPump ();
589             break;
590     }
591     return 0;
592 }
593
594 int CVICALLBACK P25ERRCallBack (int panel, int control, int event,
595         void *callbackData, int eventData1, int eventData2)
596 {
597     switch (event)
598     {
599         case EVENT_COMMIT:
600                 panel_name=panel_handle;
601                 CP=P2_comport;
602                 send_data[0] = '\0';
603                 Fmt (send_data, "%s<ERRORS?\r");
604                 PTEXT_panel_name= MAIN_STATUS_P_2;
605                 SendReadPump ();
606             break;
607     }
608     return 0;
609 }
610
611 ////////////////////////////////////////////////////////
612 // Pump #2 On.Off/Start/Stop/Reset/5errors/Set new flow rate
613 ////////////////////////////////////////////////////////
614
615
```

```c
#include <ansi_c.h>
#include <utility.h>
#include <userint.h>
#include "Quadrise MCP.h"
#include "Quadrise MCP_Declare.h"

int CVICALLBACK LogFileCallBack (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            LogFileHeader();
            break;
        case EVENT_RIGHT_CLICK:

            break;
    }
    return 0;
}


void  LogFileHeader(void)  //build based on DSN_LogFileHeader
{
    FILE *fp;
    char *DATE;
    char *TIME;

    //selecting the file to save to
    Log_File_Selected = FileSelectPopup ("", "*.log", "",
        "Enter the name of the LOG FILE", VAL_SAVE_BUTTON,
        0, 0, 1, 0, logFile);

    if (Log_File_Selected>0)    // File Selected
    {
        fp = fopen (logFile, "w");
        // Date and Time
        DATE = DateStr ();
        TIME = TimeStr ();
        fprintf( fp, "Quadrise MCP System Log File \nDate\t%s\tTime\t%s\n",DATE,TIME);
        fprintf(fp, "*****************************************************************************************************************\
n");
    }
    else    // No File Selected
    {
        // Do nothing
    }

//calibration data
    if (Log_File_Selected>0)    // File Selected
    {
    // LOGGED DATA VARIABLE NAMES

        fprintf (fp, "CS\t");       //current stage


        fprintf (fp, "P1_run\t");               //running pump #
        fprintf (fp, "P2_run\t");               //running pump #
        fprintf (fp, "P3_run\t");               //running pump #

        fprintf (fp, "TimeREF\t");      //reference time

        fprintf( fp, "TIME\t");

        fprintf( fp, "G TIME\t");


        fprintf( fp, "Flow rate P1\t");

        fprintf( fp, "Water inj P1\t");
        fprintf( fp, "Water inj P1 PV\t");

        fprintf( fp, "Water flooding\t");
        fprintf( fp, "Water flooding PV\t");

        fprintf( fp, "Pore Volume [PV]\t");

        fprintf( fp, "DP1.Permeability.water\t");
        fprintf( fp, "Delta.Permeability.water\t");

        fprintf( fp, "Flow rate P2\t");
        fprintf( fp, "Oil inj P2\t");
        fprintf( fp, "Oil inj P2 [PV]\t");

        fprintf( fp, "DP1.Permeability.oil\t");
        fprintf( fp, "Delta.Permeability.oil\t");

        fprintf( fp, "Flow rate P3\t");
        fprintf( fp, "Emulsion inj P1\t");
        fprintf( fp, "DP1.Permeability.emulsion\t");
        fprintf( fp, "Delta.Permeability.emulsion\t");


        fprintf( fp, "Pressure P1\t");
        fprintf( fp, "Pressure P2\t");
        fprintf( fp, "Pressure P3\t");

```

```
 97         fprintf( fp, "Pressure inlet\t");
 98         fprintf( fp, "Diff pressure\t");
 99
100         fprintf( fp, "Pressure differential [TP2-BPR1] , psi\t");
101
102         fprintf( fp, "M11\t");
103         fprintf( fp, "M12\t");
104         fprintf( fp, "M13\t");
105         fprintf( fp, "M14\t");
106
107         fprintf( fp, "M21\t");
108         fprintf( fp, "M22\t");
109         fprintf( fp, "M23\t");
110         fprintf( fp, "M24\t");
111
112         fprintf( fp, "M31\t");
113         fprintf( fp, "M32\t");
114         fprintf( fp, "M33\t");
115         fprintf( fp, "M34\t");
116
117         fprintf( fp, "M41\t");
118         fprintf( fp, "M42\t");
119         fprintf( fp, "M43\t");
120         fprintf( fp, "M44\t");
121
122         fprintf( fp, "BPR11\t");
123         fprintf( fp, "BPR12\t");
124         fprintf( fp, "BPR13\t");
125
126         fprintf( fp, "WATER REC\t");
127         fprintf( fp, "OIL REC\t");
128         fprintf( fp, "TEMP time\t");
129         fprintf (fp, "TEMP delta\t");
130
131         fprintf( fp, "WATER REC_ST9\t");
132         fprintf( fp, "OIL REC_ST9\t");
133
134         fprintf (fp, "TEST TUBE\t");
135
136         fprintf (fp, "Residual W_sat\t");
137
138         fprintf( fp, "WATER_EMUL_REC_ST14\t");
139         fprintf( fp, "OIL REC_ST14\t");
140
141         fprintf( fp, "EMULSION IN\t");    //Based on Mass flow meter
142         fprintf (fp, "TEMP_E delta\t");
143         fprintf( fp, "TEMP_E time\t");
144
145         fprintf( fp, "Waterflooding_temp\t");    //used for Swir calculations
146
147         fprintf( fp, "WATER_EMUL_REC_ALT\t");
148         fprintf( fp, "OIL REC_ALT\t");
149
150         fprintf( fp, "SETPOINT\t");
151
152         fprintf( fp, "CP1.1\t");
153         fprintf( fp, "CP1.2\t");
154         fprintf( fp, "CP1.3\t");
155         fprintf( fp, "CP1.4\t");
156         fprintf( fp, "CP1.5\t");
157
158         fprintf( fp, "CP2.1\t");
159         fprintf( fp, "CP2.2\t");
160         fprintf( fp, "CP2.3\t");
161         fprintf( fp, "CP2.4\t");
162         fprintf( fp, "CP2.5\t");
163
164         fprintf( fp, "CP3.1\t");
165         fprintf( fp, "CP3.2\t");
166         fprintf( fp, "CP3.3\t");
167         fprintf( fp, "CP3.4\t");
168         fprintf( fp, "CP3.5\t");
169
170         fprintf( fp, "\n");             // NEW LINE
171         fclose( fp );
172     }
173 }
174
175 //*******************************************************
176 // LogFile  : Save DATA to the log file
177 // build based on DSN_LogFile
178 //*******************************************************
179 void  LogFile(void)
180 {
181     FILE *fp;
182     //opening the file
183     fp = fopen (logFile, "a");
184     //
185 // LOGGED DATA VARIABLES - DAQ
186     //Reference time
187
188     fprintf( fp, "%i\t",stage);  //ok
189
190
191     fprintf( fp, "%i\t",pumprun_1);  //ok
192     fprintf( fp, "%i\t",pumprun_2);  //ok
193     fprintf( fp, "%i\t",pumprun_3);  //ok
```

```
194
195                 //   11 - pump #1 start 12 - pump #1 stop
196                 //   21 - pump #2 start 22 - pump #2 stop
197                 //   31 - pump #3 start 32 - pump #3 stop
198
199     fprintf( fp, "%1.4f\t",timeDAQ);  //ok
200     fprintf( fp, "%1.2f\t",step[0]);  //ok
201     fprintf( fp, "%2.4f\t", Gtime[0]);
202
203     //Flow rates and volumes injected for pumps
204
205     fprintf( fp, "%1.4f\t",PFR11[0]); //ok
206
207     fprintf( fp, "%4.4f\t",WATERINJ[0]); //ok
208     fprintf( fp, "%4.4f\t",WATERINJPV[0]); //ok
209
210     fprintf( fp, "%4.4f\t",WATERINJ2[0]); //ok
211     fprintf( fp, "%4.4f\t",WATERINJ2PV[0]); //ok
212
213     fprintf( fp, "%4.4f\t",PV[0]); //ok
214
215     fprintf( fp, "%2.8f\t",PERMWATERDP1[0]); //ok
216     fprintf( fp, "%2.8f\t",PERMWATERDELTA[0]); //ok
217
218     //oil
219     fprintf( fp, "%1.4f\t",PFR21[0]); //ok
220     fprintf( fp, "%4.4f\t",OILINJ[0]); //ok
221     fprintf( fp, "%4.4f\t",OILINJPV[0]); //ok
222
223     fprintf( fp, "%2.8f\t",PERMOILDP1[0]); //ok
224     fprintf( fp, "%2.8f\t",PERMOILDELTA[0]); //ok
225
226     //emulsion
227     fprintf( fp, "%1.4f\t",PFR31[0]); //ok
228     fprintf( fp, "%4.4f\t",EMULINJ[0]); //ok
229     fprintf( fp, "%2.8f\t",PERMEMULDP1[0]); //ok
230     fprintf( fp, "%2.8f\t",PERMEMULDELTA[0]); //ok
231
232     //Pressure from pumps
233     fprintf( fp, "%1.4f\t",P11[0]);   //ok
234     fprintf( fp, "%1.4f\t",P21[0]);   //ok
235     fprintf( fp, "%1.4f\t",P31[0]);   //ok
236
237     //Pressure inlet
238     fprintf( fp, "%4.4f\t",TP2[0]);   //ok MAX 1500.00psi
239
240     //Pressure differential
241     fprintf( fp, "%3.4f\t",DP1[0]);     //ok MAX 250.00 psi
242
243     fprintf( fp, "%4.4f\t",Delta[0]);   //ok MAX 1500 psi
244
245     //Mass flow meter #1
246     fprintf( fp, "%4.3f\t",M11[0]);   //ok
247     fprintf( fp, "%4.3f\t",M12[0]);   //ok
248     fprintf( fp, "%4.3f\t",M13[0]);   //ok
249     fprintf( fp, "%4.3f\t",M14[0]);   //ok
250
251     //Mass flow meter #2
252     fprintf( fp, "%4.3f\t",M21[0]);  //ok
253     fprintf( fp, "%4.3f\t",M22[0]);  //ok
254     fprintf( fp, "%4.3f\t",M23[0]);  //ok
255     fprintf( fp, "%4.3f\t",M24[0]);  //ok
256
257     //Mass flow meter #3
258     fprintf( fp, "%4.3f\t",M31[0]);  //ok
259     fprintf( fp, "%4.3f\t",M32[0]);  //ok
260     fprintf( fp, "%4.3f\t",M33[0]);  //ok
261     fprintf( fp, "%4.3f\t",M34[0]);  //ok
262
263     //Mass flow meter #4
264     fprintf( fp, "%4.3f\t",M41[0]);  //ok
265     fprintf( fp, "%4.3f\t",M42[0]);  //ok
266     fprintf( fp, "%4.3f\t",M43[0]);  //ok
267     fprintf( fp, "%4.3f\t",M44[0]);  //ok
268
269     //Back pressure regulator
270     fprintf( fp, "%1.4f\t",BPR11[0]); //ok
271     fprintf( fp, "%1.4f\t",BPR12[0]); //ok
272     fprintf( fp, "%1.4f\t",BPR13[0]); //ok
273
274     fprintf( fp, "%4.3f\t",WATERREC[0]);
275     fprintf( fp, "%4.3f\t",OILREC[0]);
276     fprintf( fp, "%4.3f\t",TEMPSTEP[0]);
277     fprintf( fp, "%4.3f\t",TEMPDELTA[0]);
278
279     fprintf( fp, "%4.3f\t",WATERREC_ST9[0]);
280     fprintf( fp, "%4.3f\t",OILREC_ST9[0]);
281
282     fprintf( fp, "%2.0f\t", MANUAL_TEST_TUBE[0]);
283
284     fprintf( fp, "%1.4f\t", SWI[0]);
285
286     fprintf( fp, "%4.3f\t", RECEMUL[0]);
287     fprintf( fp, "%4.3f\t", OILREC_ST14[0]);
288
289     fprintf( fp, "%4.3f\t", EMULIN[0]);
290     fprintf( fp, "%4.3f\t", TEMPDELTA E[0]);
```

```
291     fprintf( fp, "%4.3f\t", TEMPSTEP_E[0]);
292
293     fprintf( fp, "%4.3f\t", WATERFLOODINJ[0]);
294
295     fprintf( fp, "%4.3f\t", RECEMUL_ALT[0]);
296     fprintf( fp, "%4.3f\t", OILREC_ALT[0]);
297
298     fprintf( fp, "%3.2f\t", SETPOINTOUT[0]);
299
300
301     fprintf( fp, "%1.4f\t",CP11[0]);   //ok
302     fprintf( fp, "%1.4f\t",CP12[0]);   //ok
303     fprintf( fp, "%1.4f\t",CP13[0]);   //ok
304     fprintf( fp, "%1.4f\t",CP14[0]);   //ok
305     fprintf( fp, "%1.4f\t",CP15[0]);   //ok
306
307     fprintf( fp, "%1.4f\t",CP21[0]);   //ok
308     fprintf( fp, "%1.4f\t",CP22[0]);   //ok
309     fprintf( fp, "%1.4f\t",CP23[0]);   //ok
310     fprintf( fp, "%1.4f\t",CP24[0]);   //ok
311     fprintf( fp, "%1.4f\t",CP25[0]);   //ok
312
313     fprintf( fp, "%1.4f\t",CP31[0]);   //ok
314     fprintf( fp, "%1.4f\t",CP32[0]);   //ok
315     fprintf( fp, "%1.4f\t",CP33[0]);   //ok
316     fprintf( fp, "%1.4f\t",CP34[0]);   //ok
317     fprintf( fp, "%1.4f\t",CP35[0]);   //ok
318
319     fprintf( fp, "\n");         // NEW LINE
320     fclose( fp );
321 return;
322 }
323
324
325
326
```