Measurement of Regional Deformation and Volume Change of Donor Lungs during Ex Vivo Lung Perfusion Using a Stereo Vision Method

by

Jason Riley Der

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

Master of Science

Department of Mechanical Engineering

University of Alberta

Abstract

Lung transplant is a critical treatment that remains the only option for patients with end-state pulmonary illness. This treatment is underutilized because of a shortage of suitable donor lungs, leading to patients succumbing to illness while on the waitlist. Supply is limited by overly conservative rejections of possible donors. Also, transplantation is limited by hypothermic storage, the conventional preservation method, which is restricted to transplant windows of six hours. Shorter preservation periods ensure rates of primary graft dysfunction do not rise exponentially, possibly leading to recipient fatality. However, this limits transplant services to regional operations and can lead to last minute rejections as the donor lung degrade over time.

Ex-vivo lung perfusion (EVLP) preserves and monitors donor lungs at a near physiological state through mechanical ventilation, blood perfusion, and pharmaceutical treatment. The technique has the potential to improve donor lung utilization. Its measurements can accurately determine transplant viability to prevent conservative rejections. In studies, it has been shown to revitalize previously rejected donor lungs into a transplant viable condition. Also, maintaining the donor lung with mechanical ventilation and blood perfusion prevents ischemia and increases preservation periods. However, EVLP could be improved as it could host additional diagnostic sensors.

EVLP introduces the risk of ventilation induced lung injury (VILI) that could injury the lung and jeopardize transplant viability. Also, conventional EVLP diagnostic system measurements are scalar, thus are unable to differentiate the individual performance of the left and right lung, or measure asynchrony. Also, these systems would be unable to measure localized over-inflation due to heterogeneity in the donor lung's compliance.

A non-invasive camera-based processing scheme was developed to evaluate donor lung performance during EVLP treatment. The camera sensor methodology was evaluated by comparing its measurements to clinical diagnostic systems. A commercial active stereo vision system was used to measure the surface deformation of three donor lung surrogates during positive pressure mechanical ventilation at different tidal volumes. The camera system's depth measurements were used to reconstruct the lung surfaces to calculate plethysmography metrics, such as tidal volume through surface integration. Also, these metrics were derived from measurements that were simultaneously captured with a combined MEMS, or a Venturi flow rate and pressure sensor that are integrated into the clinical ventilation units. These paired measurements were used to compare the two methods, which were found to have high correlation, but poor agreement with significant systematic and proportional error relative to ventilation tidal volume. The camera-based system performed other calculations.

The camera-based system measured the left and right lung separately using image segmentation. Also, the surface deformation was scanned for peaks or troughs that would correlate with localized over-inflation and under-inflation. Peak detection was performed on surface measurements analogous to tidal volume and dynamic compliance. Lastly, the surface shape of the donor lung surrogates was characterized over one respiratory cycle by averaging all sampled cycles.

These results suggest that the camera-based method is measuring changes in donor lung surface shape with respects to respiratory cycles, however, it is currently unsuitable for plethysmography measurement. Also, a limitation of the study was that the surface deformation measurements and detected peaks were not directly validated, since the reference method did not measure regional

performance. However, the left and right lung measurements matched observations that the left lung failed to distend. Furthermore, in several cases the plethysmography systematic and proportional error of the camera method was linearly modelled with high coefficient of determination. With improvements, the method could be utilized as an additional evaluation tool to assess donor lung surface condition and health.

Acknowledgements

I am indebted to my family for their unconditional love and support. Without them, I would not have been able to accept this opportunity or persist through its challenges.

I would like to acknowledge Tevosol for co-funding this research through the Mitacs Accelerate program. Also, I will be forever grateful to their engineering team: Katie, Steve, and Calvin, for their technical guidance, mentorship, and arranging all key project experiments.

I would like to acknowledge my supervisor Dr. David Nobes for his continuous support throughout this project. Also, I would like to recognize my co-supervisor Dr. Reza Sabbagh.

Lastly, I would like to thank my fellow graduate students and the post doctorates in my lab for their camaraderie.

Table of Contents

List of Tables

List of Figures

List of Symbols

Term	Symbol	Unit
Active Stereo Vision	ASV	
Airway Pressure	P	[cmH_20]
Depth Error	Δz	[m]
Disparity Error	ΔD	[pixels]
Displacement	d	[mL]
Displacement Cycle Average	d_{cycle}	[mL]
Displacement Rate	$\dot{d}$	[mL/s]
Dynamic Compliance	$Cdyn$	[mL/cmH_20]
Expiratory Dynamic Compliance	$Cdyne$	[mL/cmH_20]
Expiratory Tidal Flow at 50% of Tidal Volume	$TEF50$	[mL/s] or [LPM]
Expiratory Tidal Volume	VTe	[mL]
Expiratory Time	tE	[s]
Ex-Vivo Lung Perfusion	EVLP	
Focal Length	f	[m]
Inspiratory Dynamic Compliance	$Cdyni$	[mL/cmH_20]
Inspiratory Expiratory Time Ratio	tIE	
Inspiratory Tidal Flow at 50% of Tidal Volume	$TIF50$	[mL/s] or [LPM]
Inspiratory Tidal Volume	VTi	[mL]
Inspiratory Time	tI	[s]
Inspiratory to Expiratory Flow Ratio	$IE50$	
Inspiratory Total Time Ratio	tITot	
Negative Pressure Ventilation	NPV	
Peak Tidal Expiratory Flow	$PTEF$	[mL/s] or [LPM]
Peak Inspiratory Pressure	PIP	[cmH_20]
Peak Tidal Inspiratory Flow	$PTIF$	[mL/s] or [LPM]
Peak Tidal Flow	PTF	[mL/s] or [LPM]
Positive End-Expiratory Pressure	$PEEP$	[cmH_20]
Positive Pressure Ventilation	PPV	
Region of Interest	ROI	
Respiratory Rate,	RR	[bpm]
Software Development Kit	SDK	
Stereo Baseline	b	[m]
Stereo Depth	z	[m]
Stereo Disparity	d	[pixels]
Stereo Vision	SV	
Surface Displacement	D	[mL]
Surface Displacement Cycle Average	D_{cycle}	[mL]
Surface Displacement Cycle Average Deviation	$D_{Deviation}$	[mL]
Tidal Volume	VT	[mL]
Time to Peak Tidal Expiratory Flow	$tPTEF$	[s]
Time to Peak Tidal Inspiratory Flow	$tPTIF$	[s]
Total Time	$tTot$	[s]
Ventilation Induced Lung Injury	VILI	

1 Introduction

1.1 Motivation

Lung transplant is a critical treatment that remains the only option for patients with end-stage pulmonary illness [1]. In 2019, the U.S. performed 2759 lung transplants with a 7.6% yearly increase in operations [2]. Although donation rates have increased, demand outstrips supply as in the same year, there were 3243 new candidates [2]. This supply-demand mismatch has led to 316 candidates succumbing to illness while on the waitlist [2]. The gap in supply and demand is partly due to the underutilization of the existing donor pool [3].

In 2019, 6.4% of procured lungs were not utilized [2] which could have saved the life of a waitlist candidate recipient. Also, 85% of potential donors are rejected, despite 41%, in a sample group, being found suitable for transplant [4]. These viable donors were erroneously rejected using a historically based evaluation method that mainly checks for contraindication such as presence of infections, donor age, and history of smoking [3],[5]. These transplant criteria have been questioned [4], and do not consider qualitative measurements of donor lung performance, which would be difficult to obtain for a donation after circulatory death (DCD). Also, donor lungs utilization is restricted by conventional hypothermic storage preservation, in which the donor lung is maintained below 4 ºC to reduce cell metabolism [6]. The donor lung is ischemic during this time which exponentially increases rates of primary graft dysfunction (PGD) past six hours [7], which could lead to transplant recipient fatality. Also, the limited preservation time causes regional heterogeneity in transplant services. To address these issues, many techniques have been explored to extend the donor pool such as using ex-vivo lung perfusion (EVLP) [6].

EVLP is a preservation technique that maintains donor lungs at a near physiological state using mechanical ventilation, blood perfusion, and pharmaceutical treatment [8]–[10]. This technique prevents ischemia, reducing the risk of PGD, and has been shown to have longer preservation times than hypothermic storage. During preservation, lung performance is monitored [11]–[13] allowing more accurate transplant viability evaluations that could prevent conservative rejections. Also, EVLP has been shown to revitalize previously rejected lungs [14]–[16], improving their health into a transplant viable condition. Therefore, EVLP could expand the

existing donor pool and improving patient outcomes. However, the system could be improved, as it has been predicted to be a treatment platform that could host additional diagnostic sensors [8].

One limitation of EVLP is that it introduces the risk of ventilation induced lung injury (VILI) [17], jeopardizing transplant viability and patient well-being. Conventional plethysmography and pressure measurements such as tidal volume and positive expiratory-end pressure (*PEEP*) are used to predict VILI [18], [19]. However, VILI can be caused by excessive stretching or tearing from improper ventilation and exposure to high concentrations of oxygen [20]. Ultimately these method are not capable of measuring the real mechanism of VILI, lung stress and strain [17], [18], [21]–[23]. Another limitation of conventional EVLP diagnostic systems is that measurements are scalar. Therefore, they are unable to differentiate the individual performance of the left and right lung, thus are unable to measure asynchrony. Also, these systems would be unable to identify the location of localized over-inflation due to heterogeneity in the donor lung's compliance. To address these issues, an additional diagnostic sensor could be added to the EVLP device to measure full field physical distension, strain, or stress of the donor lung.

1.2 Hypothesis

A diagnostic sensor for measuring full field performance of a donor lung inside an EVLP device would have several design constraints. For example, the system can not interfere with the EVLP device or invasively contact the donor lung which would jeopardize transplant viability. Also, an ideal system would provide real-time measurements with accuracy and repeatability equal or greater than the existing EVLP plethysmography system. Lastly, it would be beneficial if the technological basis was well established so it could be more easily approved by the FDA.

There are several existing plethysmography techniques that could be implemented for this application. Optical reflectance (OR) motion analysis [24], [25] and respiratory inductive plethysmography [26] could provide full field stress and strain of the donor lung. However, they are entirely invasive, as reflective markers or sensors would be attached directly to the donor lung. Non-contact alternatives such as MRI and MDCT [27], [28] could interfere with the EVLP device, and is computationally and financially expensive, even when disregarding feasibility of the size of the system. Lastly, respirometers such as spirometers and pneumotachograph [29] are

essentially already implemented in the EVLP device. The methods so far are either invasive or would interfere with the EVLP device. However, there have been imaging techniques that have been successfully tested on in-vivo patients. For example, digital image correlation (DIC) was used to non-invasively measure the full field stress and strain of an in-vivo heart, during an open-heart surgery [30], [31]. Also, structured light plethysmography (SLP) was used to measure chest wall deformation of a patient to calculate tidal parameters [32]–[34], which was found to be comparable to clinical measurement techniques [35]. These are some examples of photogrammetry techniques that could be used to monitor the donor lung during EVLP [36].

Photogrammetry, the measurement of physical objects from images [37], is a good candidate for improving EVLP diagnostics. Imaging techniques require few pieces of equipment, often they use one or multiple cameras, optionally with either a light projector or surface markers. They can measure the shape of an environment non-invasively such as stereo vision (SV) and structured light 3D scanning (SL) [38], [39]. Also, shape measurements can be used to derive other metrics such as strain and stress when using DIC, plethysmography with SLP, and volume measurement [40], [41] . These measurements can be obtained in real-time for certain approaches such as SV and SLP [38], [42]. However, other techniques must post-process images for measurements like DIC. Other limitations are mainly imposed by their processing scheme and available computational processing power, rather than hardware. For example, cheap commercial webcams can be used for SV [43] and do-it-yourself SL when used with a laser pointer [44]. However, camera quality and resolution still directly impact accurate and precision. Lastly, a camera-based systems are software flexible since multiple photogrammetry processing schemes can utilize the same type of images. For example, color images could be processed using Eulerian video magnification to measure blood flow [45]–[47], while simultaneously using scene flow to measure a 3D displacement field [48]. Therefore, photogrammetry offers several development benefits and non-invasive options to measure donor lung performance during EVLP.

Photogrammetry techniques were reviewed to select a technique for this research project. Several techniques were considered, mainly DIC, SLP, and SV. DIC is ideal for this application because it provides full-field deformation, strain, and stress measurements. The technique requires multiple high-resolution cameras and a speckle pattern on the object of interest to track its

surface position and deformation. However, these requirements are not ideal in this application. DIC is practically limited to small FOV scales, and its measurement require a computationally intensive post-processing method, making real-time results difficult to achieve [49]. Lastly, while the surface markers were acceptable in the study for open heart surgery [30], [31], obtaining FDA approval to apply markers on a donor lung may still prove difficult.

Initially, SLP seemed promising for this application. The technique non-invasively provides real-time plethysmography and a surface model that could be used to identify localized performance. However, the EVLP optically transparent chamber cover would interfere with the SLP calibrated structured light pattern, deviating the light pattern in the image from its calibrated state. As a result, SLP would render invalid measurements of the donor lung when imaging through the EVLP cover.

Lastly, SV was considered and selected for this application. The method has been proven reliable in various fields, including medicine for radiotherapy targeting [50], anthropometry [51], [52], and laparoscopy in-vivo measurement [53]. It only requires two cameras, provides real-time depth or shape measurement, and several commercial research and development platforms are available for a reasonable price. Also, the plethysmography processing techniques used in SLP and OEP could be adapted for SV measurements. Also, the optical distortions caused by the EVLP cover could be corrected for using existing methods [54], [55]. The only drawback is that it is not capable of strain and stress measurement or tracking surface stretching or shear.

Hypothesis Statement:

It is hypothesized that a stereo vision system could be used as an additional diagnostic sensor for monitoring a donor lung during EVLP.

Ideally, the system would provide real-time surface shape measurements of the donor lung for whole lung and regional plethysmography, and to detect localized over-inflation. This research project will evaluate the potential benefits and identify possible obstacles of implementing this SV based diagnostic system.

1.3 Objectives

The objective of this research project is to develop an SV based diagnostic system to measure surface deformation of a donor lung during EVLP treatment. The development process will include technique and hardware selection, design of a data acquisition protocol, and outline a data processing scheme. Also, the research project should evaluate the accuracy and repeatability of the developed system. Lastly, experiments with donor lungs or surrogates should be performed to quantify the system's measurement behavior. In summary, the scope of the research project is as follows:

1. Review the fundamentals of plethysmography of the EVLP.
2. Review the fundamentals and limitations of stereo vision.
3. Select and review a commerically available imaging system as the diagnostic sensor.
4. Develop a processing scheme for plethysmography and regional surface deformation measurement.
5. Evaluate the stereo vision diagnostic system experimentally with donor lung surrogates using method comparison analysis with a clinical ventilation measurement system.
6. Summarize findings from experiment results and provide recommendations.

Notably, some practical issues such as obtaining valid measurements through the EVLP transparent cover are not included in the scope of this research. The research project will focus on identifying the potential benefits and obstacles of an SV based diagnostic system, rather than developing a system ready for implementation. Also, the project scope will focus on the EVLP device configuration developed by the company Tevosol, since they co-funded and technically supported this research.

2 Plethysmography Fundamentals

Plethysmography is the measurement of volume change in the body. In this case, an objective of this research is to perform pulmonary plethsymosgraphy, or measure lung volume change during a respiratory cycle or breath, which is measured during EVLP. Conventionally, pulmonary plethysmography is performed using respirometry that uses airflow rate and pressure sensors. This technique is used to calculate metrics such as tidal volume and dynamic compliance to evaluate the performance of the donor lung. In this chapter, plethysmography metrics are reviewed to define measurements obtained during EVLP, which the ASV method should be able to measure. The following approach is most applicable to ASV since it is based on SLP.

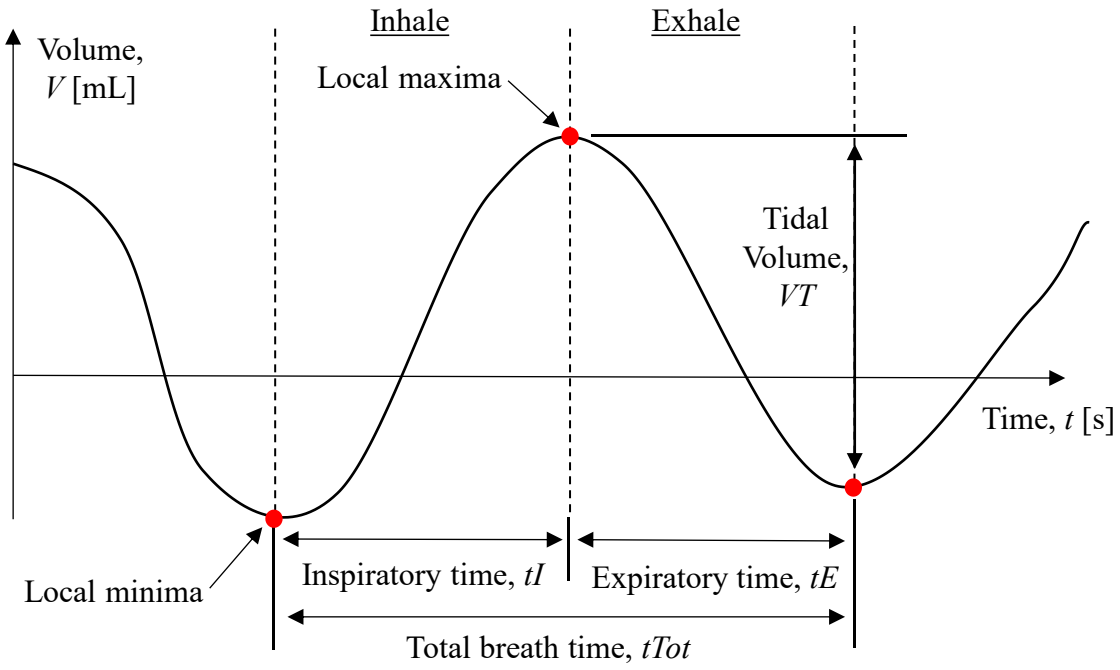


Figure 2.1. Schematic of volume signal segmentation and metrics

Local maxima and minina points in the volume signal are used to identify the beginning and end of each respiratory cycle, inhale, and exhale phase, as seen in Figure 2.1. Tidal volumes were found using these local extrema points. Tidal volume is the change in volume of the lung, measured as the volume of air inhaled or exhaled during a breath. Also, the timing characteristics of each respiratory cycle [35], [56], or breath, can be measured such as inspiratory time, *tI*,

expiratory time, *tE*, and total respiratory cycle time, *tTot*. Typically, these metrics are used to obtain respiratory rate, *RR*, as the inverse of the *tTot*, and the ratios *tI* / *tTot* and *tI* / *tE* [35].

$$Cdyn = \frac{\Delta V}{\Delta P} = \frac{VT}{PIP - PEEP} \tag{2-1}$$

If given pressure measurements, tidal volume could be used to find dynamic compliance, which is the dynamic elasticity of the lung. In (3-7), *PIP* and *PEEP* are the peak inspiratory and end-expiratory pressures within a respiratory cycle [19]. They can be found at the local extrema points of the pressure signal.

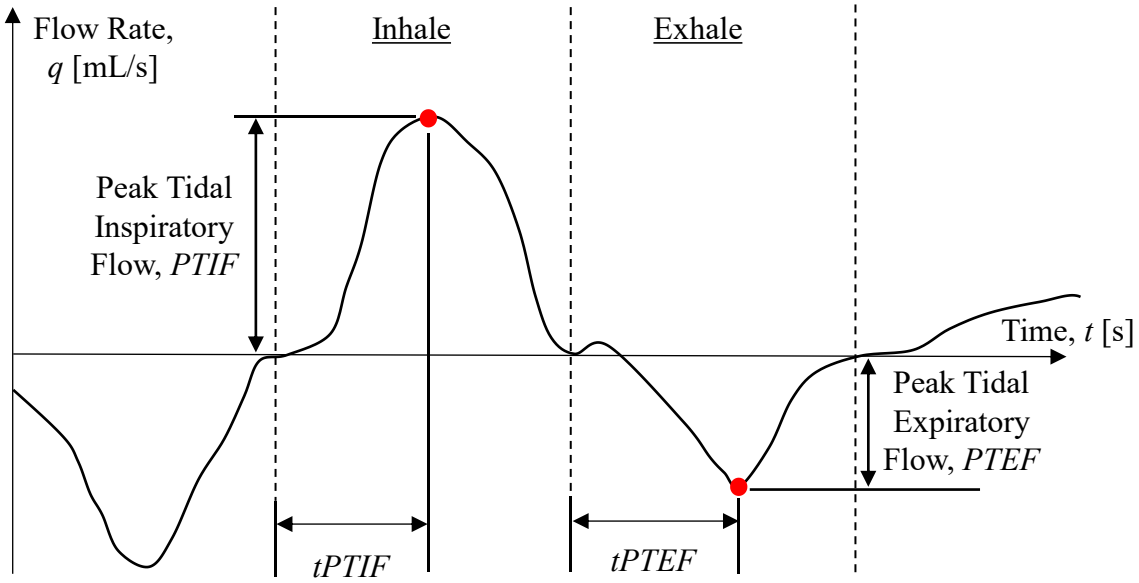


Figure 2.2. Schematic of flow rate signal segmentation and metrics

Similar metrics can be taken from the flow rate signal, as seen in Figure 2.2. The peak tidal inspiratory and expiratory flow, *PTIF* and *PTEF*, are found as the peak flow values within the inhale and exhale phases. The time to these points, *tPTIF* and *tPTEF*, are taken with respects to the start of the inhale and exhale phases [35].

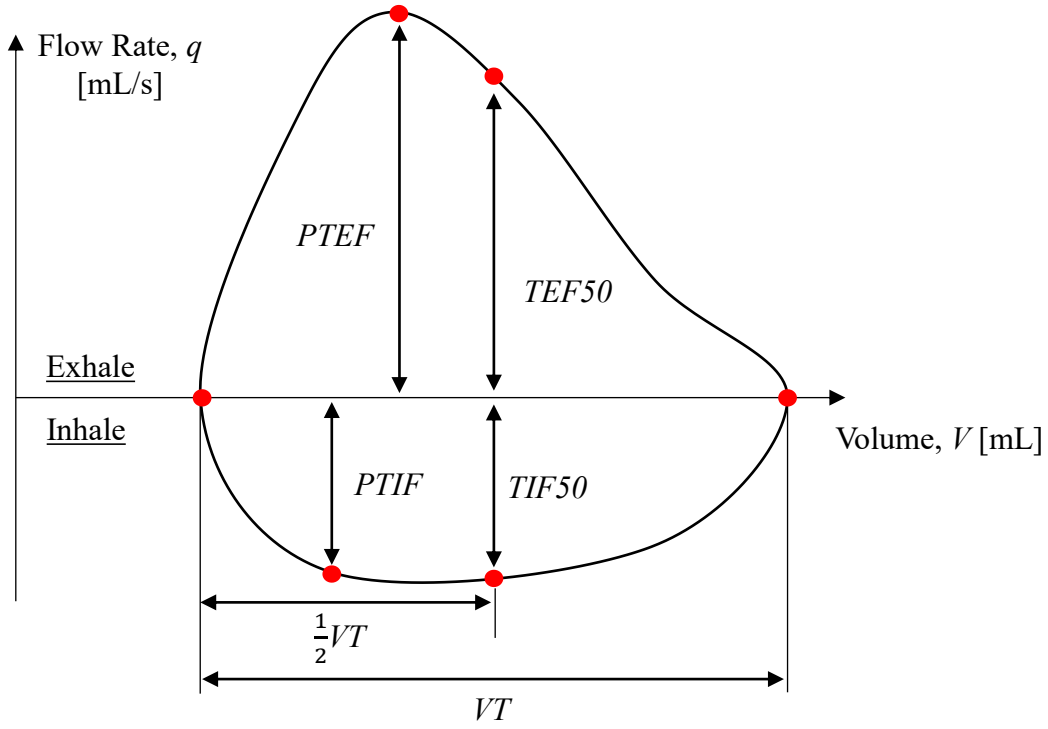


Figure 2.3. Schematic of a respiratory cycle Lissajous curve flow-loop of volume and flow rate

The performance of a patient, or donor lung, can be visually characterized by a respiratory cycle flow-loop, or Lissajous curve, that plots volume versus flow rate, as seen in Figure 2.3 [35]. Also, defining metrics are measured such as inspiratory and expiratory tidal flow at 50% of tidal volume, *TIF50* and *TEF50*, and the inspiratory to expiratory flow ratio, *IE50* = *TIF50* / *TEF50* [35].

3 Fundamentals of Active Stereo Vision

In 1960, Larry Roberts first discussed the extraction of 3D measurements from images, a technique now known as photogrammetry [37]. Since then, photogrammetry has been an active field of research, as many techniques have been developed using an assortment of camera configurations and algorithms [57]. Stereo vision is one of the more established techniques in this fields.

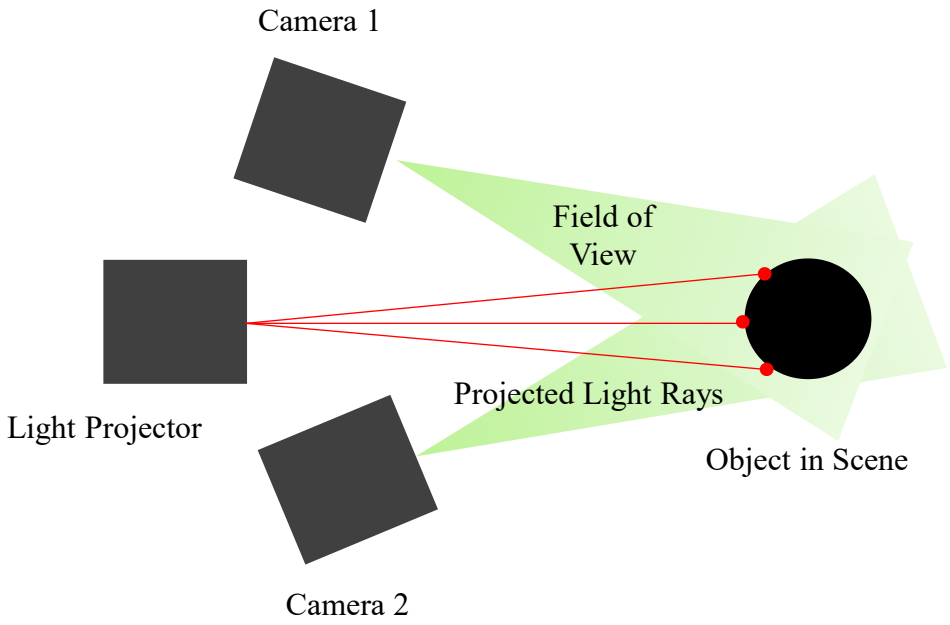


Figure 3.1. Schematic of the components for active stereo vision

Stereo vision is a depth estimation technique that uses two cameras to imitate stereopsis, as seen in Figure 3.1 [58]. There are four fundamental steps: calibration, rectification, digital image matching, and triangulation. Calibration finds the parameters needed to model the relationship between an image and the physical world. Rectification and digital image matching simplify and solves the correspondence problem, which is finding matching pixels in both camera images that correspond to the same point in the real world. Triangulation uses the difference in matching pixel positions, or disparity, to calculate the distance of the object. This process generates a 2D image that encodes the distance of a scene, called a depth map. Approaches to improve stereo

vision performance have been investigated, which includes using a light projector to enhance the digital image matching step, called active stereo vision [59]. In this chapter, the fundamentals of active stereo vision are reviewed to better understand the limitations and strengths of the technique.

3.1 ASV Modelling

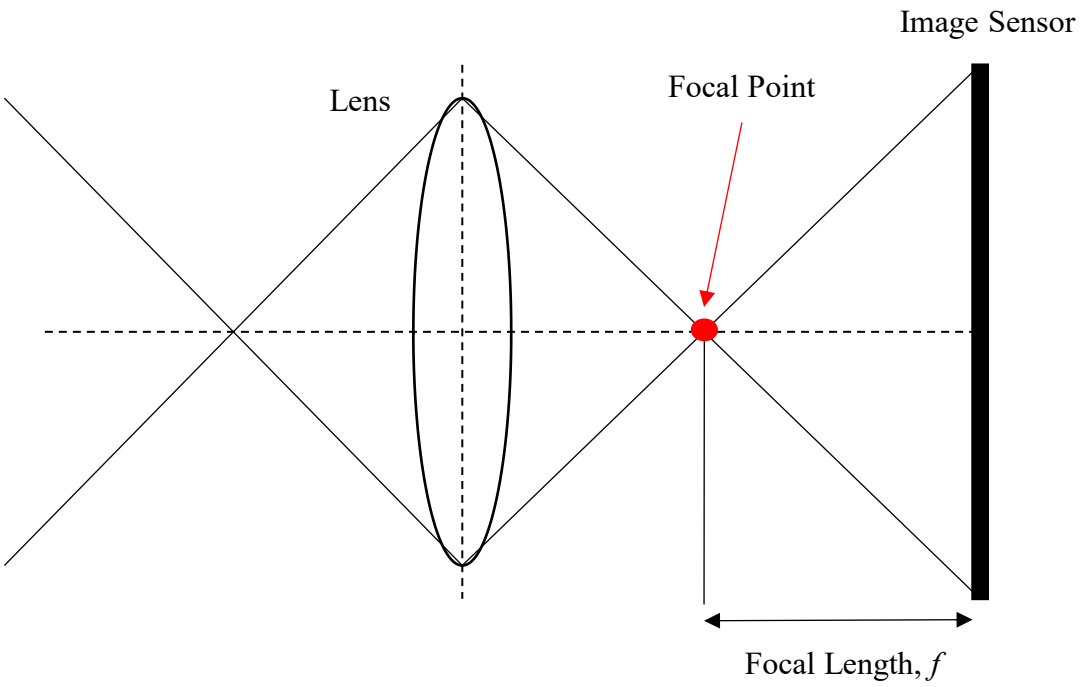


Figure 3.2. Schematic of a simplified camera

A camera has two main components: a lens, or series of lenses that forms an optical system, and an image sensor, often a CCD or CMOS, as seen in Figure 3.2. The optical system focuses light reflected from the environment to the image sensor, converting the light into a discrete digital signal to form an image. Light is focused through a single point, called the focal point, a known distance from the image sensor, the focal length, *f*. To perform photogrammetry, this camera system must be modelled to relate the pixels in an image and the physical world, commonly achieved using the pinhole model [60].

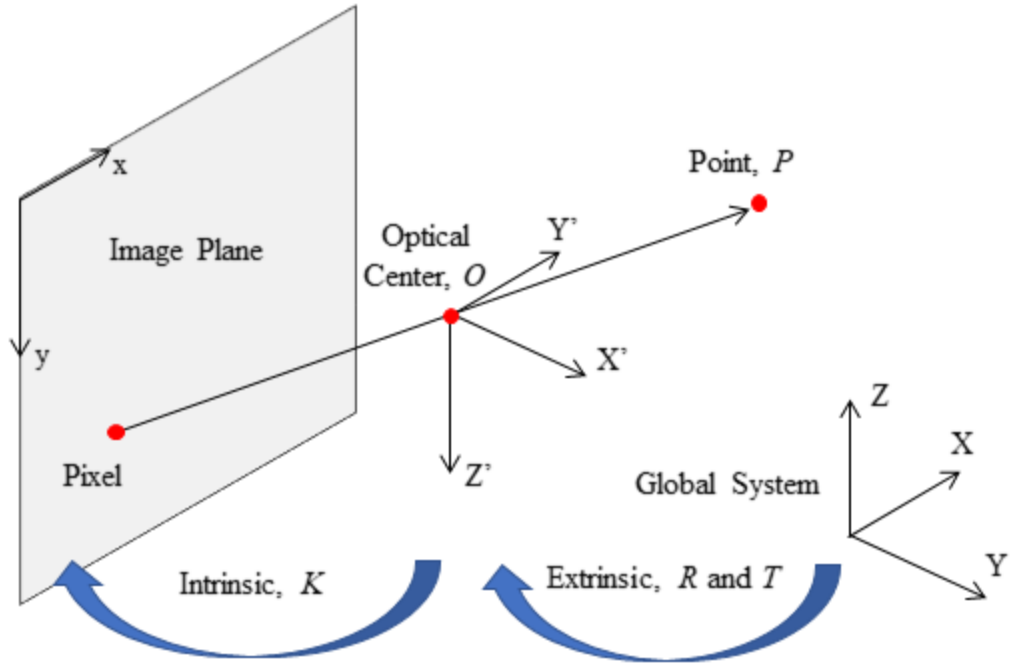


Figure 3.3 Schematic of the pinhole model and relationship between coordinate systems.

The pinhole model simplifies a camera into an image plane for the CCD sensor, and an optical center for the focal point, as seen in Figure 3.3. Light reflecting off a point in physical space is modelled as a ray that intersects the optical center and image plane. This relationship projects the point in the physical global system onto the image plane [61], [62]. The pinhole model is defined by intrinsic and extrinsic parameters that allow a 3D point to be represented in different coordinate systems, which is defined as:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = KRT \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad (3\text{-}1)$$

where
x, y, z are the coordinates of the pixel in the image coordinate system
X, Y, Z are the coordinates of the point in the physical coordinate system

In the pinhole model (3-1), K is the intrinsic matrix, and the rotation matrix, R, and the translation matrix, T, are the extrinsic parameters that give position and orientation. The vectors on both sides of equation (3-1) are augmented to perform a homogenous matrix transformation.

Notably, all points lie on the image plane, so z is equal to one. The five intrinsic parameters define the position of the image plane relative to the optical center [63].

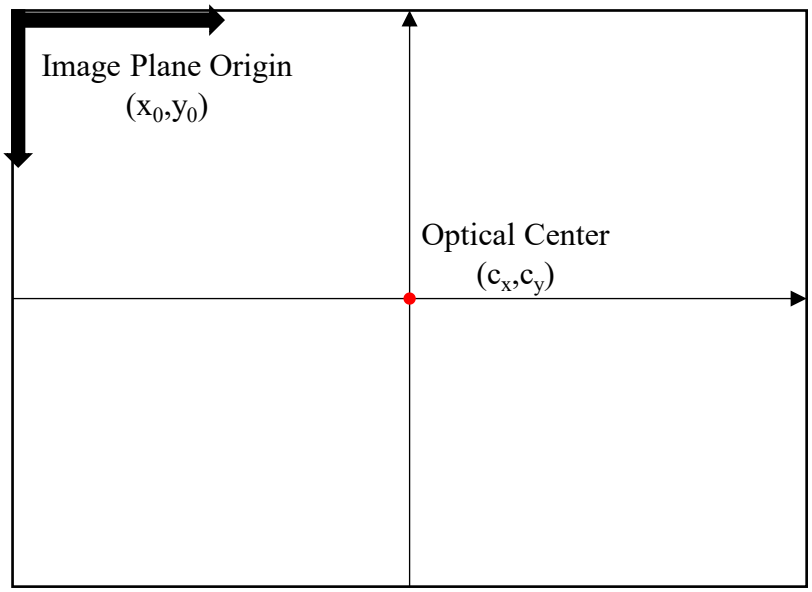


Figure 3.4. Schematic of the optical center intrinsic camera parameter

The position of the optical center, c_x and c_y, are two of the intrinsic parameters as seen in Figure 3.4. Both parameters are measured in pixels and are typically half the image resolution [64].

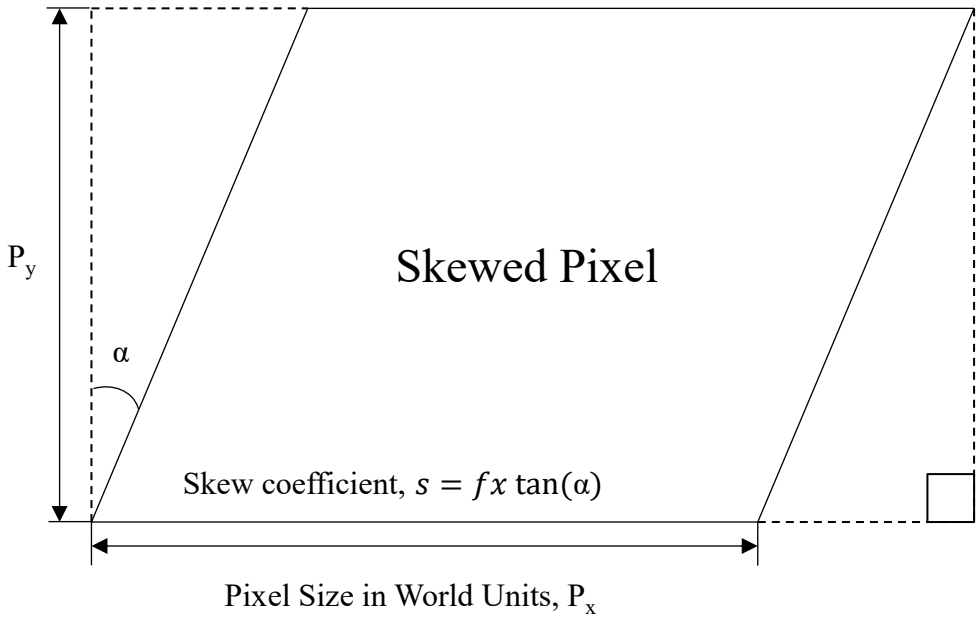


Figure 3.5. Schematic of the skew coefficient intrinsic camera parameter

The skew coefficient, *s*, measures the angle of skew when the pixels are not perfectly perpendicular, as seen in Figure 3.5. Typically, the pixels are perfect rectangles where *s* is zero [64].

The K matrix is defined as:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}2)$$

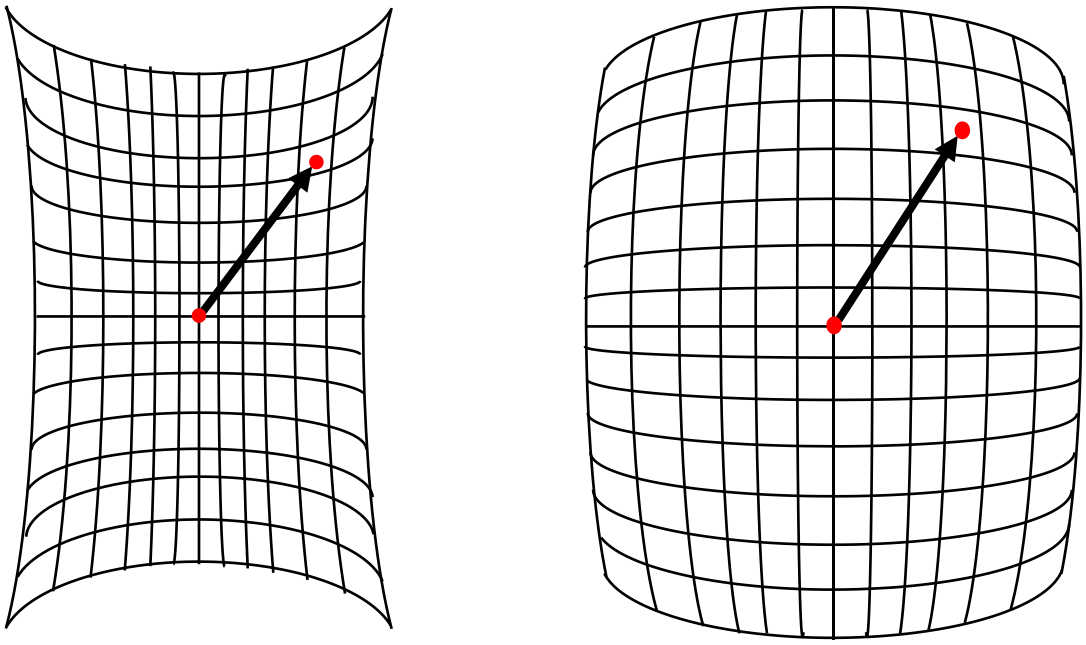
where
f_x is the focal length in the *x* direction
f_y is the focal length in the *y* direction

Five intrinsic parameters, seen in (3-2), define the relationship between the image coordinate system and the focal point. In an ideal pinhole model, the f_x and f_y are equal [64]..

$$RT = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}3)$$

The 12 extrinsic parameters in (3-3) describe the physical orientation and position of the focal point relative to the global coordinate system. In a stereo vision configuration, the extrinsic parameters will define the position and orientation of one camera relative to the other camera, which will be the origin for depth measurements [64].

The 17 intrinsic and extrinsic parameters do not account for optical, alignment, and manufacturing imperfections, which cause distortions and modelling errors. There are four common types of distortion: radial, tangential, thin prism, and total distortion [64], [65],[61].



(a) *(b)*

Figure 3.6. Schematic of (a) negative radial distortion or barrel distortion and (b) positive radial distortion or pin cushion distortion

Radial distortion is image magnification that depends on radial distance from the optical center. Pincushion and barrel distortion, as seen in Figure 3.6, are respectively caused by positive and negative radial distortion [61], [66].

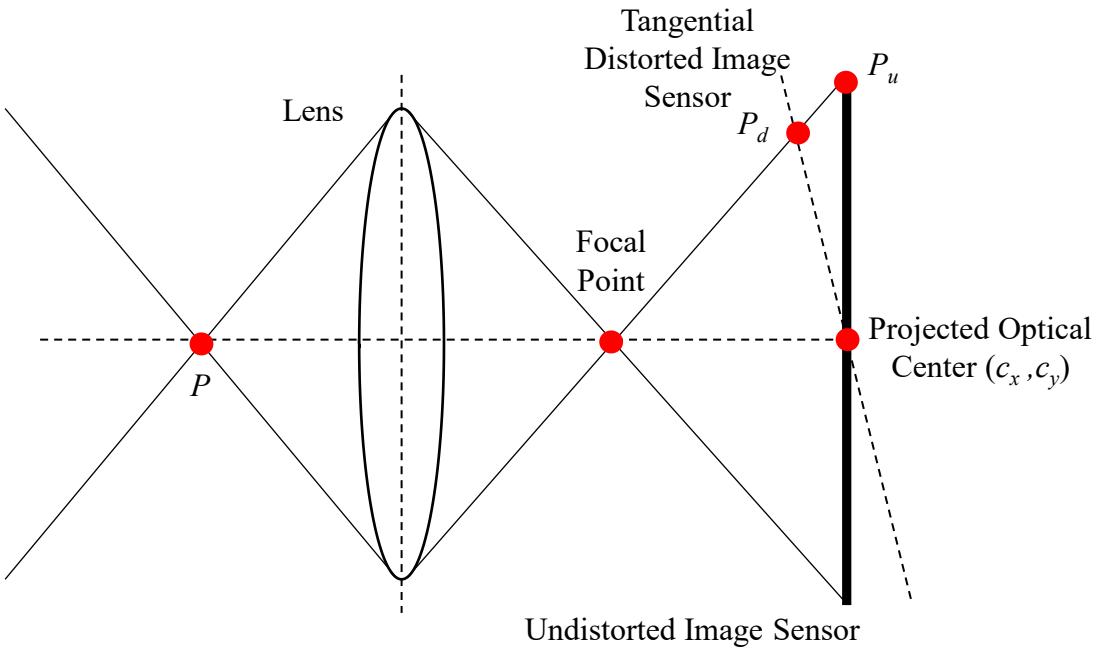


Figure 3.7. Schematic of tangential distortion

Tangential distortion is cause by the lens and image plane not being parallel, as seen in Figure 3.7. Thin prism distortion is caused by manufacturing imperfections of the lens. Total distortion is the summation of the three previously listed types of distortion [61], [66].

3.2 ASV Calibration

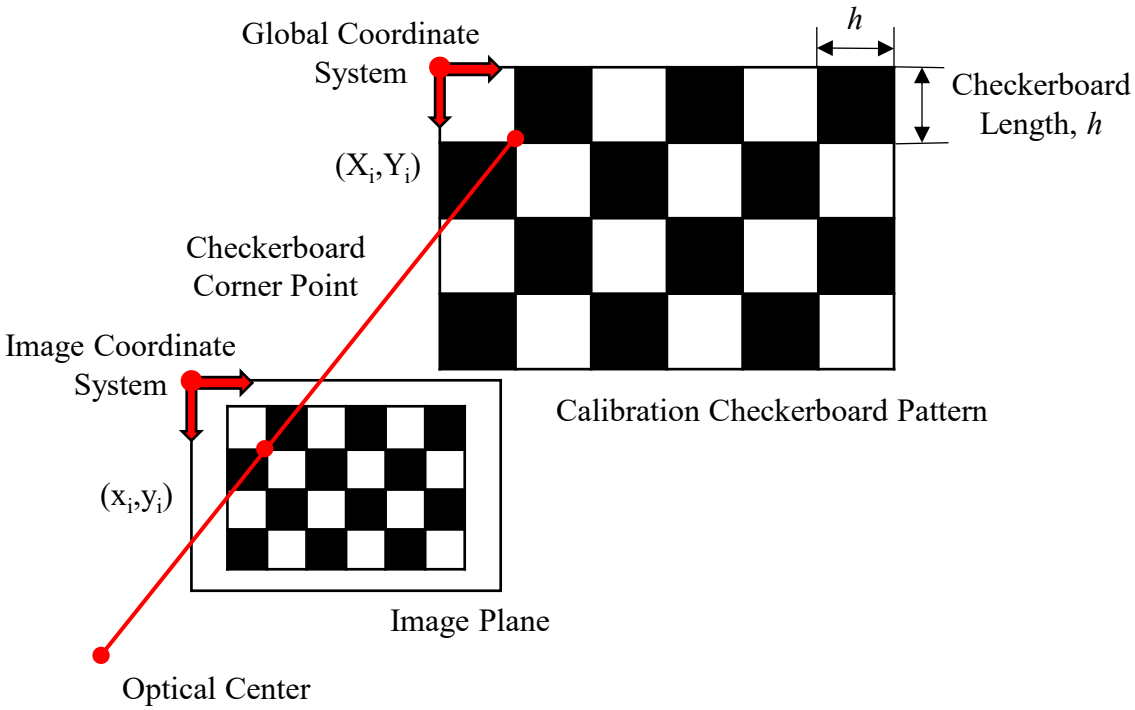


Figure 3.8. Schematic of checkerboard pattern calibration

Calibration is the process of solving for camera intrinsic and extrinsic parameters, along with the distortion coefficients. Typically, photogrammetric calibration is performed, in which images are taken of an object with a geometric pattern and known dimensions, in various positions and orientations. Commonly the object is a rectangular checkerboard pattern of black and white squares, as seen in Figure 3.8. The calibration images are used to solve for the camera parameters, by fitting the position of the checkerboard corners, acting as feature points, in the image to the real world using the known geometry of the pattern [57], [67], [68].

There are several calibration algorithms that commonly use the error between the projected 2D images and the known 3D calibration feature points. There are three main types of calibration algorithm [64], [65], [68], [69]:

1. Iterative non-linear optimization
2. Closed form linear
3. Hybrid, two-stage

Iterative non-linear calibration methods iteratively minimize a cost function to solve for the camera parameters and distortions. Typically, the cost function depends on the error between the model's current prediction and the known 3D location of feature points identified in the calibration scene. This approach requires a good initial guess and sufficient iterations to reach convergence, but usually provide a good result [65].

Closed form calibration methods directly solve the camera matrix. For example, the Hall and Faugeras-Toscani methods [65] compute the camera parameters based on the least-squares between the projected 2D image points and known 3D points. This approach is simple and fast to implement. However, it is less accurate than iterative methods and does not consider lens distortion [65].

Two-stage calibration methods use both the iterative and closed form approaches to solve different camera parameters. Typically, the closed form method is used to obtain a good initial guess for the iterative non-linear which almost guarantees convergence [65].

These methods can be used to calibrate any camera configuration, including stereo vision systems, in which the extrinsic parameters relate the position of the two cameras.

3.3 Image Pair Rectification

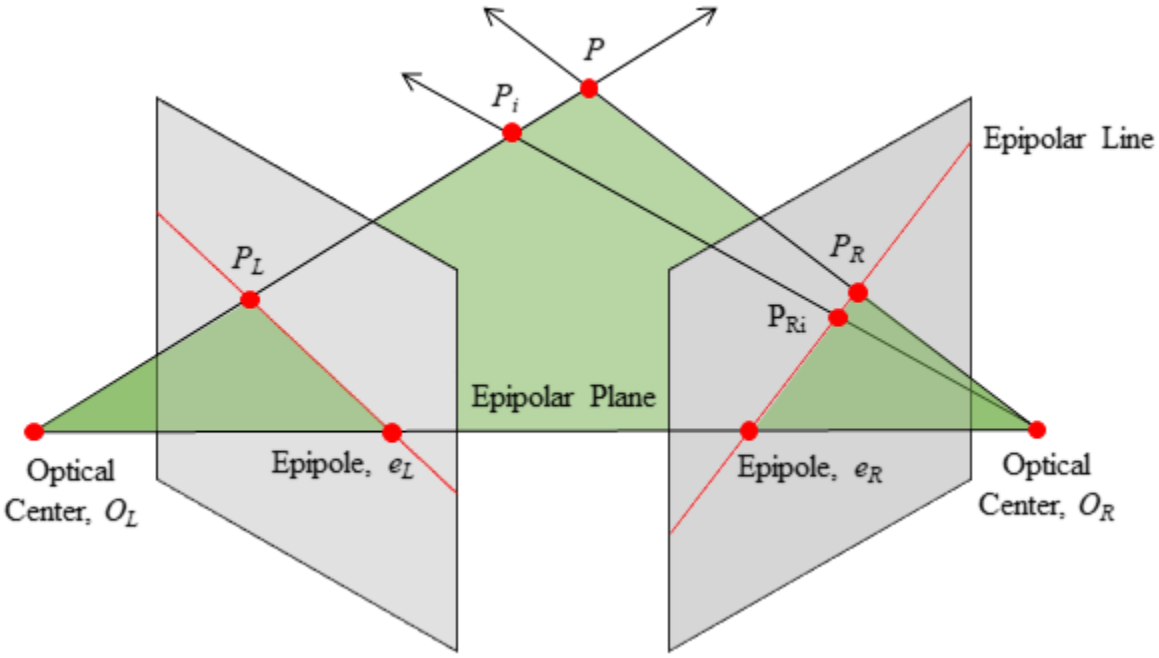


Figure 3.9 Schematic of epipolar geometry

Image rectification projects images from two cameras onto the same plane to improve the speed of digital image matching. This step is not required for stereo vision, however, it is common because it is impractical to have a camera configuration with perfect co-planarity [67], [70]–[74]. The method is derived from epipolar geometry, as seen in Figure 3.9.

Epipolar geometry describes a situation where two cameras see the same 3D point, P. A plane forms between the optical centers of the left, O_L, and right, O_R, cameras and the 3D point, which is called the epipolar plane. The projections of the optical centers and the 3D point onto the left and right image planes lie on this epipolar plane. These points include the epipoles, e_L and e_R are the projections of the optical centers onto the other camera's image plane. Also, the projections of the 3D point onto the left, P_L, and right, P_R, image planes lie on the projected line of the epipolar plane. This projected line, called the epipolar line, intersects the epipolar plane, image plane, and the projected points. This is true for all points along the epipolar plane such as P_{Ri}.

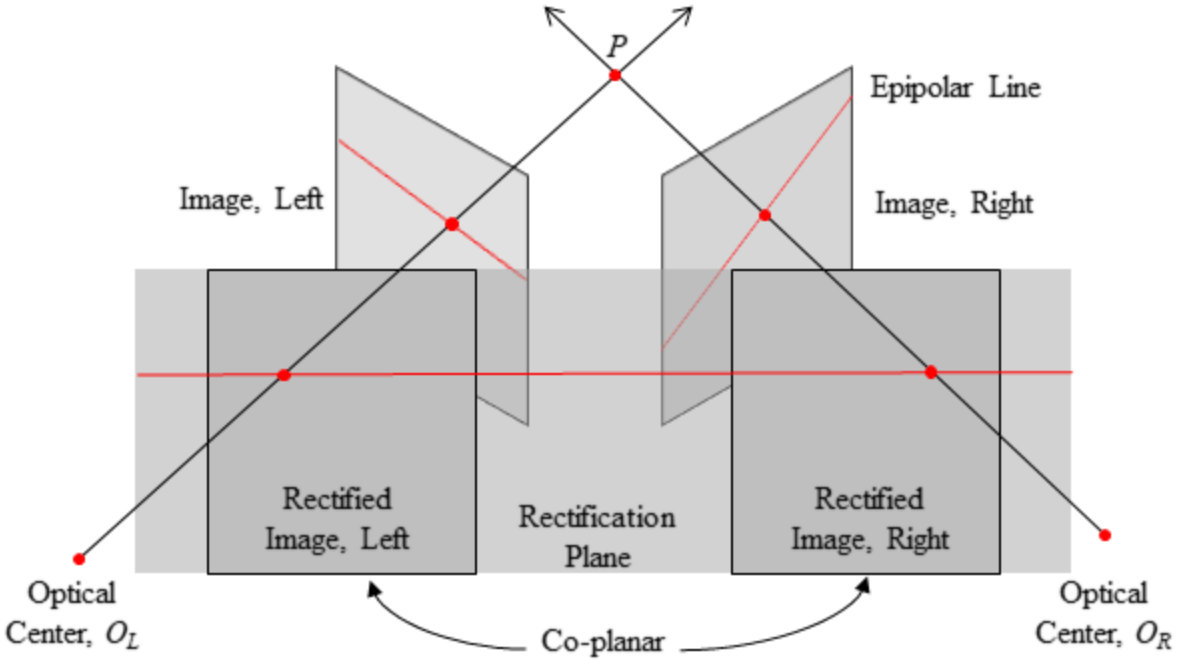


Figure 3.10. Schematic of image rectification

Rectification uses the epipolar geometry relationship to define a linear algebra transformation that can be solved to vertically align the epipolar lines of the images by projecting the images onto the same plane, as seen in Figure 3.10. The rectified images allow the digital image matching step to find matching pixels only by horizontally scanning the same image row, instead of the original skewed epipolar lines [67], [70], [75].

3.4 Digital Image Matching

Digital image matching solves where parts of an image correspond to another image, which is the correspondence problem. It is arguably the most important step of stereo vision, as it is the most non-trivial problem, and directly influences the depth estimation accuracy [76].

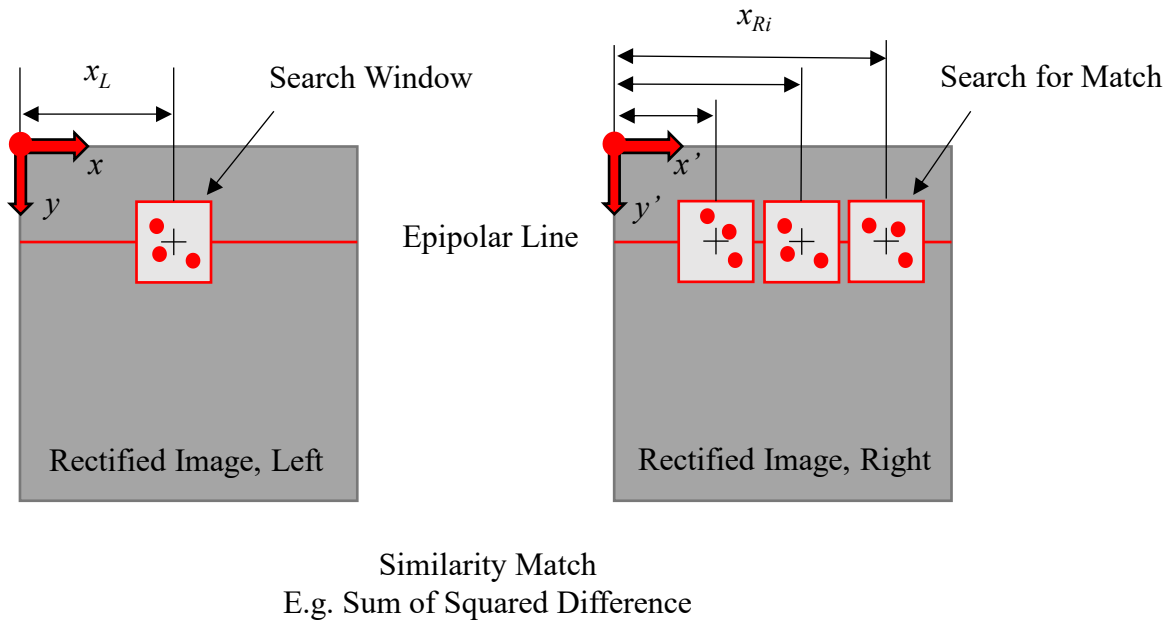


Figure 3.11. Schematic of digital image matching correspondence search

Ideally, digital image matching would find the correspondence between individual pixels. However, this is nearly impossible because they do not provide enough robust information. Instead, matching is performed between sub-regions, or search windows, to include the information of neighboring pixels, as seen in Figure 3.11. The corresponding match can be found by either searching the entire image, or only searching near the original's location [77]–[79]. A match can be found using a line search, along the epipolar line, if the image pair has been rectified [75], [80].

A match is determined by measuring the similarity between the original and candidate windows, using metrics such as sum of squared differences using:

$$s = \sum_{(x,y)\epsilon I} (I_1[x,y] - I_2[x,y])^2 \qquad (3\text{-}4)$$

where I_1 and I_2 are the pixel intensity of the left and right image of the windows. Correspondence is decided by the highest similarity. Since similarity depends on absolute pixel intensity, the method is sensitive to distortions, noise, and illumination. Also, featureless, and low contrast scenes lead to poor or failed matches.

Alternative matching algorithms perform cost function minimization on the similarity metrics [81], or feature detection such as least squares matching, which are more robust to distortions and feature orientation.

Regardless of the matching algorithm, they all calculate the difference in pixel position between corresponding matches, to generate a disparity map. The disparity map is used as an input for depth estimation using triangulation.

3.5 Projected Light Pattern

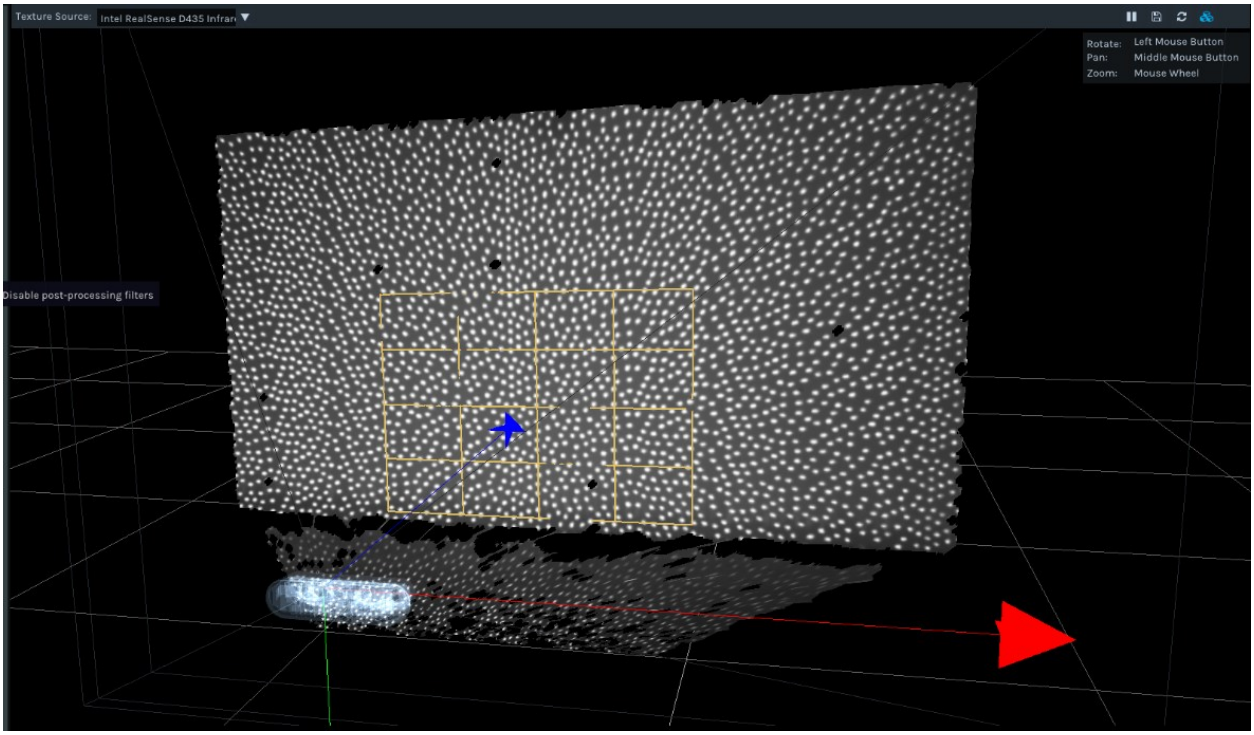


Figure 3.12. Image of active stereo vision with projected light pattern in the scene

Active stereo vision uses a projected light pattern to create features and increase contrast in the scene to improve image matching, as seen in Figure 3.12. The projection can either be a speckle pattern, or a calibrated structured light. This allows active stereo vision to overcome common problems such as curved or smoothed surfaces, called the false boundary problem [80]. Stereo vision without the projector, is also known as passive stereo vision.

3.6 **Depth Estimation Triangulation**

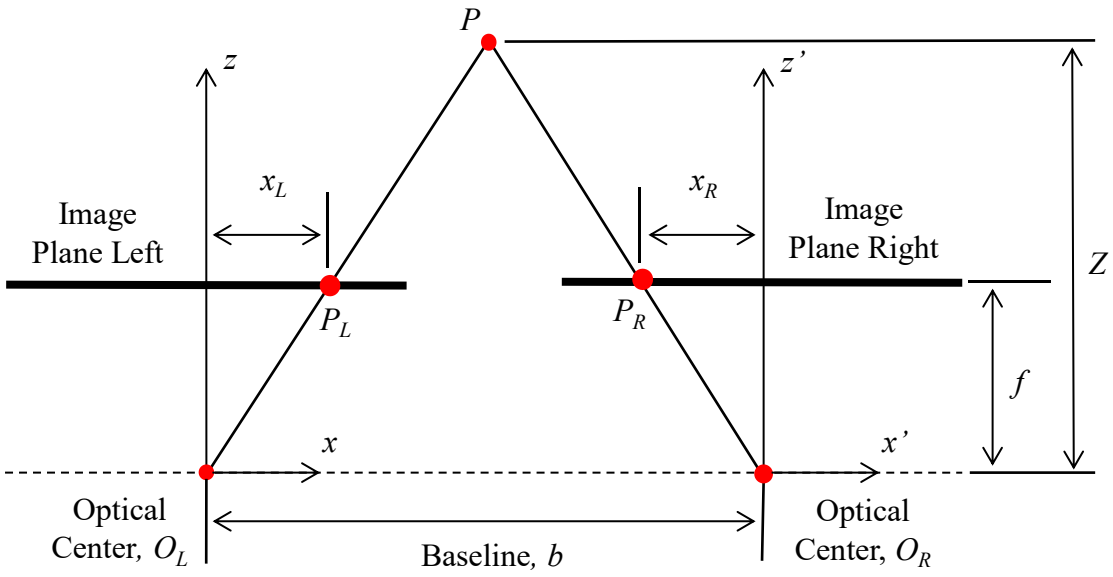


Figure 3.13. Schematic of stereo vision triangulation

Triangulation calculates the distance, Z, of 3D points, P, using trivial geometry and the pixel disparity found using digital image matching, as seen in Figure 3.13 [70]. The 3D point intersects the left and right image planes at P_L and P_R, which respectively have pixel x coordinates x_L and x_R in their own coordinate system. The optical centers of the left, O_L, and right, O_R, are horizontally spaced by a baseline, b, distance, and their camera axis align. In the simplest cases, the cameras have the same focal length, f.

$$\frac{Z-f}{Z} = \frac{b-(x_L - x_R)}{b} \tag{3-5}$$

The triangulation equation is derived using similar triangles. The denominators of (3-5) are from the triangle formed by O_L, O_R, and P, while the nominators are from the triangle formed by P_L, P_R, and P. Notably, x_R is negative because of the sign convention of the right camera's coordinate system for x'.

$$Z = \frac{fb}{(x_L - x_R)} = \frac{fb}{d} \tag{3-6}$$

Equation (3-6) is obtained from (3-5) after solving for the depth, Z, in the left camera relative to the optical center of the left camera, O_L. The focal length, f, and baseline, b, are found from calibration, while the disparity, d, is found from digital image matching.

While stereo vision is a powerful technique for depth estimation, it has limitations as it requires non-trivial modelling of the camera system, calibration, and matching. Also, the technique dependents on having a direct line of sight by both cameras, and various lighting phenomena can trick the matching algorithm.

3.7 Stereo Vision Limitations

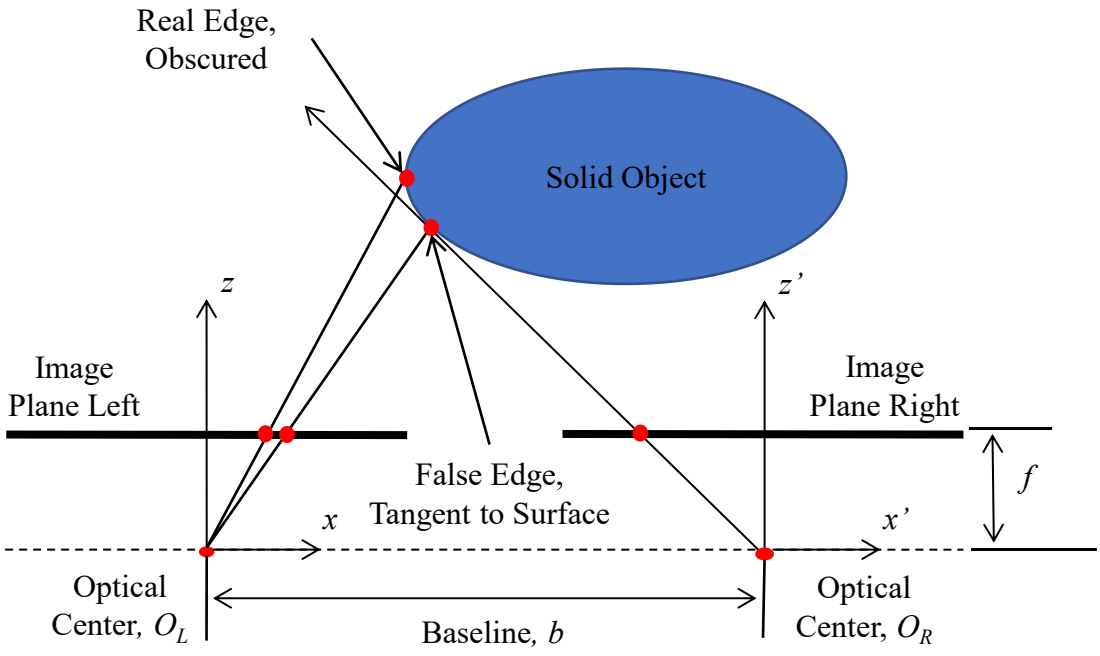


Figure 3.14. Schematic of the false boundary problem from curved surface

There are several well-known limitations with passive and active stereo vision that cause invalid depth measurement. Since the technique is dependent on pixel intensity to determine matches, it is sensitive to the reflectivity of surfaces, environmental illumination, and any image distortion

such as noise [82]. For example, a smooth reflective surface with uniform lighting will have no global or local features for the matching algorithm to solve the correspondence problem, called matching ambiguity [83]. Another example is when an edge in light intensity is created by lighting conditions instead of a physical edge. This situation causes a false edge to be detected, called the false boundary problem [77], [82]. Also, a false can be detected when the edge of an object can be obscured around a curved surface, for one camera as seen in Figure 3.13. ASV position and orientation can cause other issues, such as occlusion, which is when a surface is not visible to both cameras [83]. In this case, the depth map will have a hole at this location, which is a pixel without a depth measurement [83].

$$\Delta z = \frac{z^2}{b * f} \Delta D \tag{3-7}$$

where
z is depth
Δz is depth error
b is baseline
f is focal length
ΔD is disparity error

Depth estimation error is derived from (3-6). Equation (3-7), shows that depth error increases quadratically with distance and linearly with disparity error [84], [85]. Therefore, depth error is reduced when measurements are taken at a close range, and the matching algorithm disparity error directly contributes to depth error.

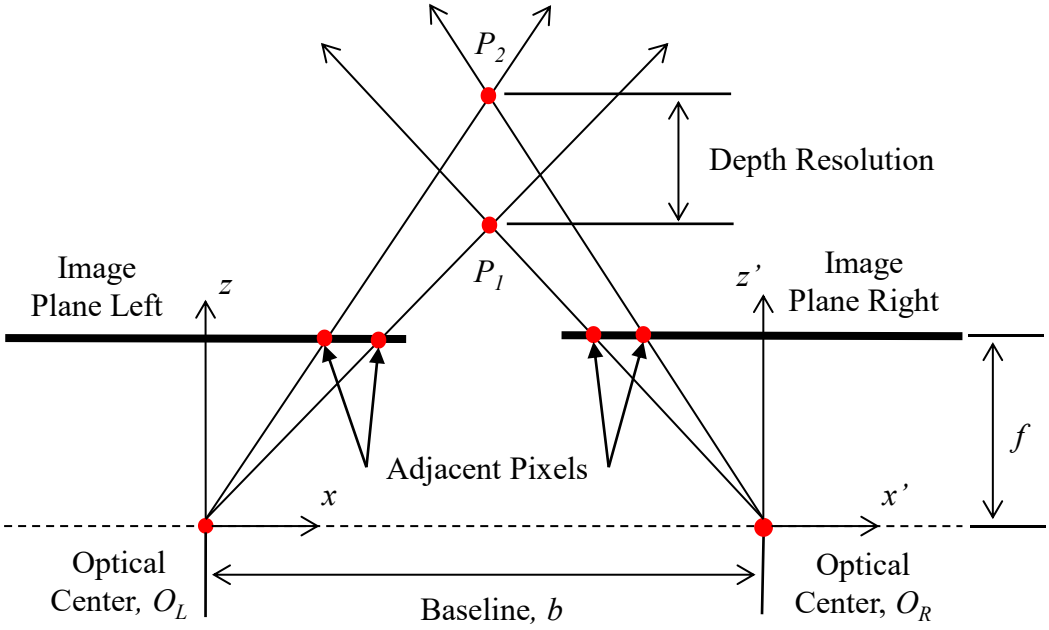


Figure 3.15. Schematic of depth resolution

The depth resolution is determined by distance, sensor resolution, and baseline as seen in Figure 3.15 [52]. Depth uncertainty, like depth resolution, is non-trivial to model, thus is typically obtained experimentally [86]–[89], [90]. Thus, depth resolution is finer at closer ranges, with a larger baseline and focal length, and a higher sensor resolution.

3.8 Fundamentals of Active Stereo Vision Conclusion

This chapter reviewed the components, mechanics, and limitations of stereo vision. Stereo vision is a photogrammetry technique for depth estimation. In its simplest form, passive stereo vision uses two aligned cameras, with the same focal length, that are horizontally separated by a baseline distance. Active stereo vision improves this system configuration with a light projector to add a non-invasive speckle pattern or calibrated structured pattern to the scene. The pinhole model is used to relate images to the physical world, which requires calibration to obtain intrinsic and extrinsic parameters. The correspondence problem is solved using a global or local area based matching algorithm, simplified to a line search by rectification, to find the disparity between matching pixels. Triangulation uses the camera parameters and disparity to estimate the

distance of objects in a scene. Depth error quadratically increases with distance, and linearly with disparity error. Depth resolution is dependent on the camera configuration baseline, focal length, sensor resolution, and distance.

4 Intel RealSense D435 Active Stereo Vision Platform

The Intel RealSense D435 active stereo-vision system was selected for this project, after surveying commercially available vision systems. It was qualitatively determined to be the best development platform for this application, because it fit the technical requirements of the project. The depth accuracy, field of view, and effective range were suitable to measure a donor lung inside an EVLP device. Also, the Intel RealSense D435 was the only surveyed system with software tools and usability features supported by the manufacturer [91], [92].

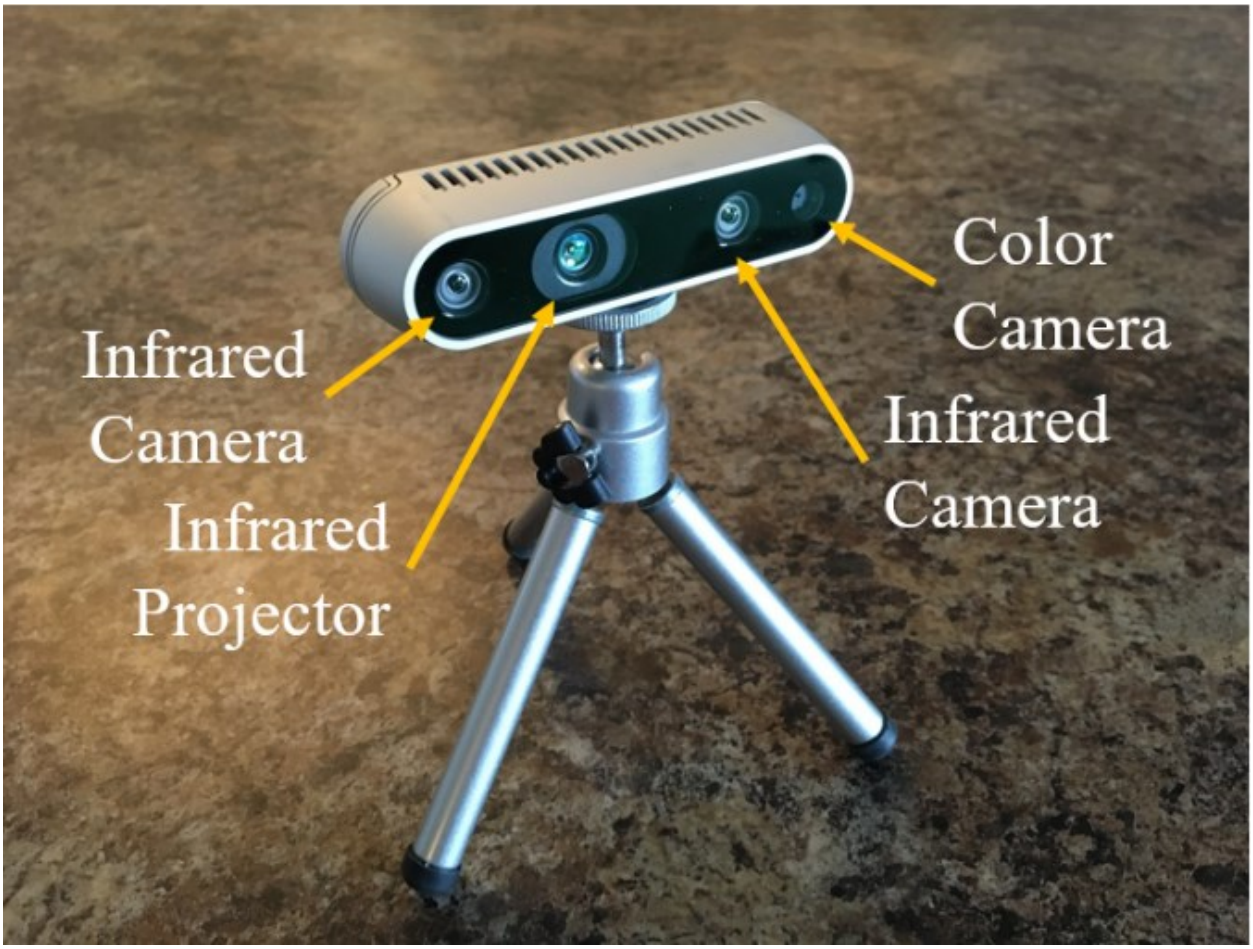


Figure 4.1. Annotated image of the Intel RealSense D435

The Intel RealSense D435 performs ASV using a depth sensor, composed of two infrared imagers and projector and has an auxiliary color camera, as seen in Figure 4.1. Notably, it has a USB 3.0 port and threaded mounting holes for a tripod and machine screws, shown in the mechanical drawing in Appendix A. An onboard depth processing unit provides real-time depth

maps at the same frame rate as the imagers. The infrared, color and depth images can be accessed at different resolutions and frame rates, up to 1920 × 1080 and 90 fps, limited by the transfer rate of the selected USB cable. The generated depth maps provide measurements in a 87° × 58° × 95° field of view with a depth error below 2% within two meters [93]. All data and configurations can be accessed programmatically using provided Intel Realsense software tools.

The entire Intel Realsense camera series has access to a software development kit (SDK) with MATLAB compatibility using a C++ wrapper that provides access to low level device functions. The SDK is documented on several sources such as GitHub, whitepapers, and its own dedicated website [92]. Also, the documentation provides example scripts that were used as templates to programmatically access images and set camera configurations using MATLAB. Notably, there is an active forum community of developers and customers that supports troubleshooting.

The SDK can be used to develop a custom application for the Intel RealSense D435. The manufacturer provides functional example applications to interface with the Intel RealSense D435. They were used throughout this research project to calibrate the device camera parameters, evaluate its depth accuracy [94], [95], and perform data acquisition.

4.1 Calibration and Depth Evaluation

Figure 4.2. Screenshot of the Intel RealSense Dynamic Calibration application

The Intel RealSense D435 intrinsic camera parameters are calibrated by the manufacturer using a high precision calibration target. As a safety feature, the intrinsic properties are typically not configurable since they are assumed non-mutable. However, the extrinsic parameters are expected to change when the individual cameras shift position and orientation inside the case. Therefore, the Intel Realsense Dynamic Calibration application [96] calibrates for the infrared and color cameras' extrinsic parameters using images of a calibration pattern, as seen in Figure 4.2. In this project, the pattern was printed onto a standard paper page and glued to a sheet of acrylic. Alternatively, Intel offers a mobile phone application that displays a calibration target [96].

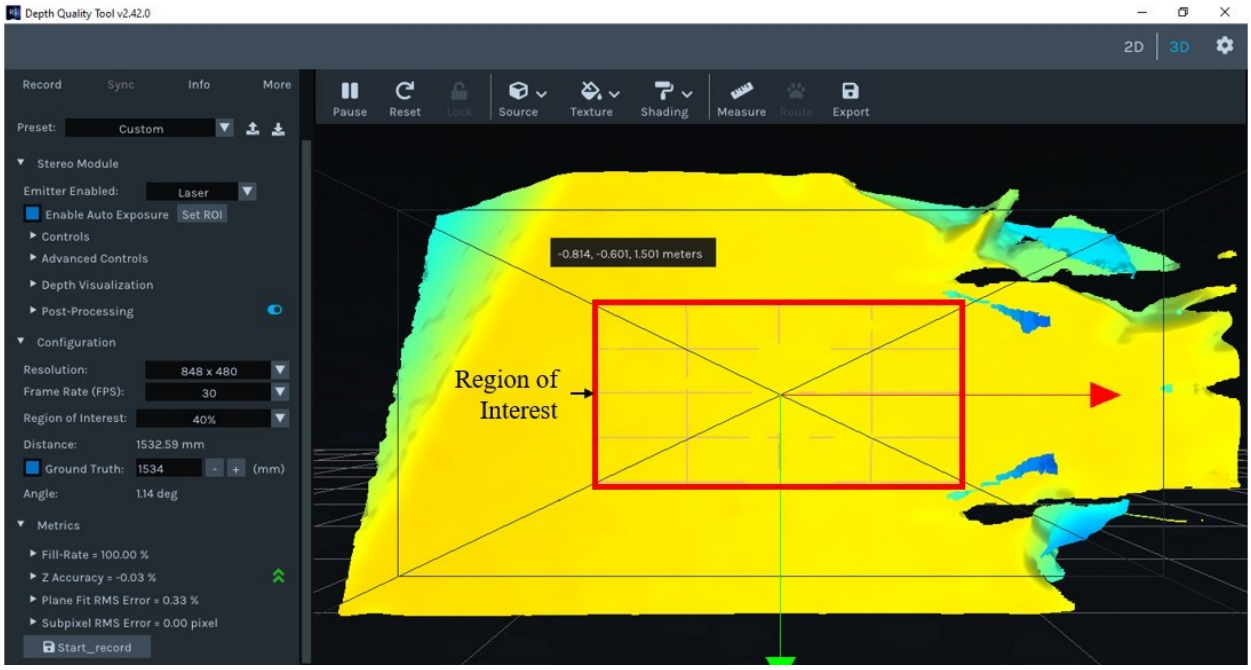


Figure 4.3. Screenshot of the Intel RealSense Depth Quality Tool

The calibrated extrinsic parameters were evaluated by measuring the overall depth quality of the Intel RealSense D435 using the Intel Realsense Depth Quality Tool. The depth map measurements of a flat wall, at a known distance and within a region of interest (ROI), as seen in Figure 4.3, were used to calculate five depth quality metrics. If they do not fall within recommended values, then the manufacturer suggests re-calibrating the device. The five depth quality metrics are:

1. Z-Accuracy
2. Fill Rate
3. RMS Error
4. Temporal Noise
5. Sub-Pixel Error

Before calculating the metrics, the ROI depth measurements are rotated to align with a fitted plane, to correct for the relative orientation of the camera to the flat surface. The *z*-accuracy is the percentage error between the median of the aligned ROI depth measurements and the ground truth. The fill rate is a percentage of pixels with a valid depth measurement, non-zero and not an outlier. The root mean square (RMS) error is calculated between the rotated depth measurements

and the nearest point on the fitted plane. The temporal noise is the variance in depth measurement between frames. Lastly, the sub-pixel error is estimated based on the RMS error [94]. Notably, the depth measurements are relative to the depth origin of the Intel RealSense D435 that is located at the front of the left infrared camera [93].

4.2 Data Acquisition

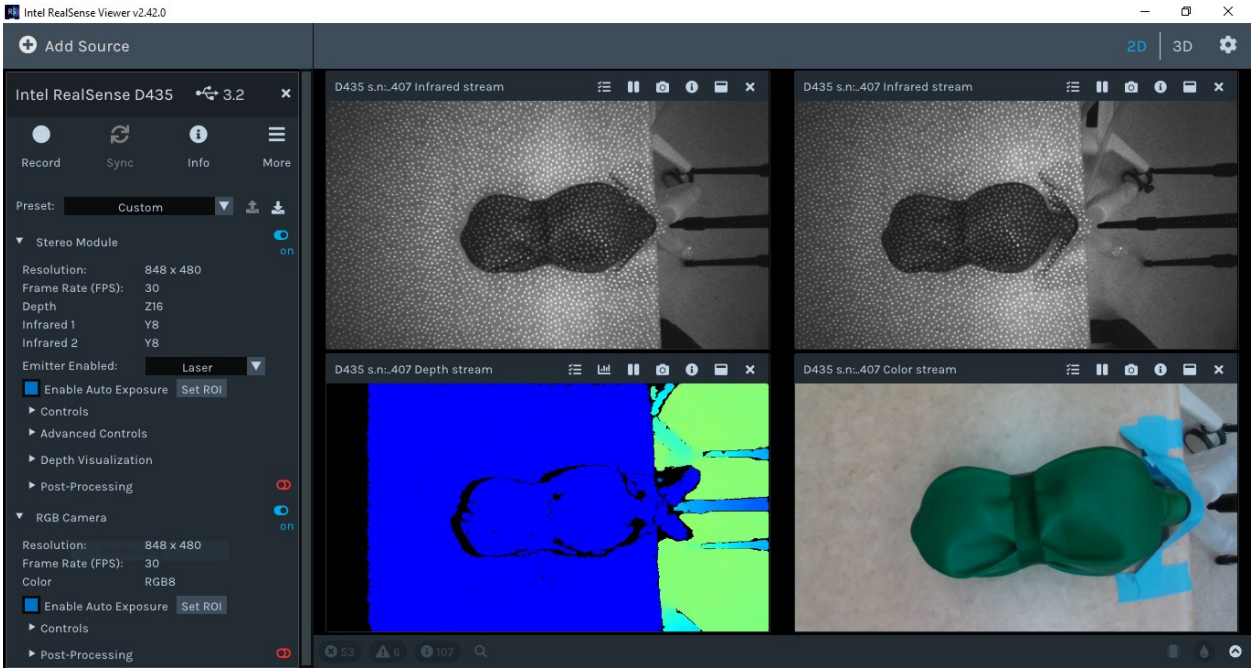


Figure 4.4. Screenshot of the Intel RealSense Viewer application

The Intel Realsense Viewer application, as seen in Figure 4.4, was used to interface with the Intel RealSense D435 through a laptop and USB 3.0 cable. The top two images in Figure 4.4 are from the left and right infrared cameras, from left to right. The bottom left image is the depth map generated from active stereo vision. The bottom right image is the color image from the Intel RealSense D435 auxiliary color camera. The application uses the SDK to configure the device, including resolution and fps, and save data to a ROS bag file that is accessed in MATLAB [97].

The MATLAB ROS toolbox was used to read the Intel Realsense Viewer files for images and metadata such as timestamps. The images are post-processed using the SDK to synthetically replay the videos and apply fundamental image filters for temporal and spatial smoothing [97], [98]. Also, the SDK can perform projective transformation on the depth maps to align them with

the color images [97], [98]. These steps are performed using the custom function rs2ReadRosbag() that can be found in Appendix B. The post-processed images and metadata are saved to mat files for future processing.

4.3 Intel RealSense D435 Conclusion

The Intel RealSense D435 active stereo vision system was selected for this research project because it met all technical and financial requirements. Also, it comes with software tools such as a software development kit (SDK) and pre-made applications for camera calibration, depth evaluation, and data acquisition. The SDK was used to read videos recorded from the Intel RealSense D435 into MATLAB for future post-processing to measure donor lung performance.

5 Deformation and Plethysmography Processing Scheme

A processing scheme was developed to measure whole lung plethysmography and regional surface deformation of a donor lung using depth maps from active stereo vision. In this research project, the method was implemented using MATLAB toolboxes for image and point cloud processing. Also, it was improved over a series of experiments. However, the fundamental steps remained the same and should be applicable to other photogrammetry techniques. The general approach has five steps:

1. Stereo Vision to Acquire Depth Maps
2. Image Segmentation to find the Lungs from Depth Maps
3. Deprojection of Segmented Lung from Depth Maps
4. Surface Reconstruction from the Point Clouds to create Surface Maps
5. Measure Surface Maps for Global and Regional Plethysmography Metrics

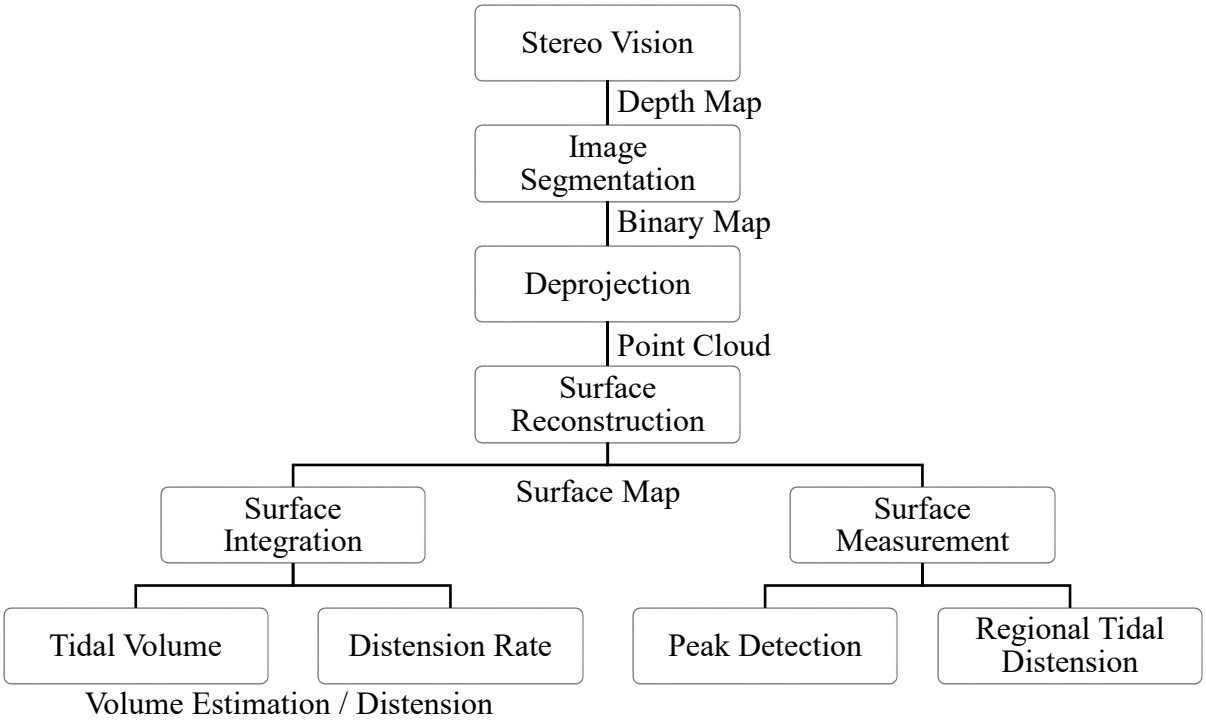


Figure 5.1. Flow chart of the processing scheme

These steps are summarized by the flow chart in Figure 5.1. This chapter explains the problems and possible solutions of each step. In this research project, the depth maps were acquired using the method described in Chapter 4.2 .

5.1 Image Segmentation

Image segmentation is the process of identifying region(s) of interest inside an image. Regions must be homogeneous, continuous, and do not overlap, but their union covers the entire image. These regions are defined in a labelling image, or binary map, with the same dimensions as the segmented image [99]. The pixels belonging to a given region are labelled a logical true, while outside the region are logical false. These conditions are summarized in five rules [99], [100]:

1. Union of all partitions is the entire region of the image
2. Regions are continuous
3. Regions are disjoint
4. Similar pixels are of the same region
5. Pixels that are not similar must not be of the same region

In this project, image segmentation was used to select only the depth measurements of the donor lung in the depth maps. This was achieved by segmenting any of the aligned images such as the depth, left infrared, or color images since their binary maps are interchangeable. Several established segmentation methods were explored, which are broadly categorized into edge-based and region-based methods [101].

5.1.1 Edge-Based Segmentation

The edge-based methods identify the edges of objects, as borders of regions, to partition an image. Often, these methods are sensitive to noise and need to find an appropriate edge threshold, which is a non-trivial problem, however, they are computationally simple and fast. Typically, these methods have three main steps [100]:

1. Edge Detection
2. Edge Linking
3. Edge Localization / Thinning

Edge detection is the process of finding edges as abrupt changes in pixel intensity using first or second derivative operators, called edge detectors, creating gradient images [100], [102]. For digital functions, such as images, derivatives are approximated using finite Taylor's Series expansions [100]. Edges are detected by binarizing the gradient images through thresholding.

An example of edge detection is the Canny edge detection method that uses a first derivative of Gaussian operator to find the gradient of an image. This is followed by a step called non-maximal suppression that smooths the gradient image to remove noise. Finally, two thresholds are used to identify "strong" and "weak" edges in the gradient image, which is called hysteresis thresholding [100], [101], [103].

Edge linking ensures edges are continuous by linking neighboring edges in a binary map. Most methods only link edges that have similar gradient magnitude and or direction [100], [104].

Edge localization or thinning ensures the edges are one pixel thick [100]. This can be achieved using morphological operations such as skeletonization or erosion [100], [105].

Edges could be found in depth maps based on differences in height. Alternatively, edges could be found by distinguishing the lung from the background by differences in color.

5.1.2 Region-Based Segmentation

Region-based segmentation methods identify and grow regions, and eventually define region boundaries once dissimilar regions meet [101]. In most methods, some pixels are assigned to continuous regions, while many pixels are left unlabeled. Pixels neighboring labelled regions are measured for similarity [106] to determine if they similar enough to belong with that region, or if a region border should be formed when they are dissimilar. Also, two regions may merge if their region similarity is above a threshold. These methods are robust to noise but cost more time and memory since they often involve iterative steps [95].

Lazy snapping is an example of a region-based segmentation method developed by Microsoft Research Asia [107]. It is an interactive graph cut method intended for color images, similar in function to Photoshop's Magnetic Lasso tool. The method has four interactive and automatic steps [107].

1. Automatic pre-segmentation to improve computational efficiency
2. Interactive foreground and background seed region labelling
3. Automatic region growing through Gibbs Energy minimization
4. Interactive boundary editing for pixel accurate boundaries

In the first step, the image is segmented into many regions. In this state the image is unsuitable for depth map segmentation. However, it allows the lazy snapping method to merge these groups instead of growing regions from pixels. In the second step, the user interactive draws roughly where the foreground and background are in the image, labelling these pixels as seed regions. The third step is uses Gibbs Energy to measure similarity between neighboring pixels, grow regions, and define region borders. Lastly, the user is prompted to add or remove pixels along the region boundaries to manually refine the results [107]. This last step is not implemented by the MATLAB function.

The watershed method is another region-based segmentation method that requires seed locations and iteratively grows from the seed regions. The method is likened to the topographical concept of a watershed and basin, where all the water in a region flows towards a regional minima. The segmentation method identifies a watershed region for each seed location that is defined, which includes the background. The boundaries between the watershed regions are called watershed lines. The metric of similarity is based on the geodesic distance that measures the relative intensity of the segmented image, similar to the height in topographic maps [108]–[110].

5.2 Deprojection

Deprojection is the process of transforming an image into 3D points [89], [111], which was used to obtain a collection of 3D points, called a point cloud, of the surface of the lung from a segmented depth map. Since each depth map pixel and 3D point are paired, information such as color and region label can be inherited by the point. After some processing, the point clouds can be used to measure displacement of the lung.

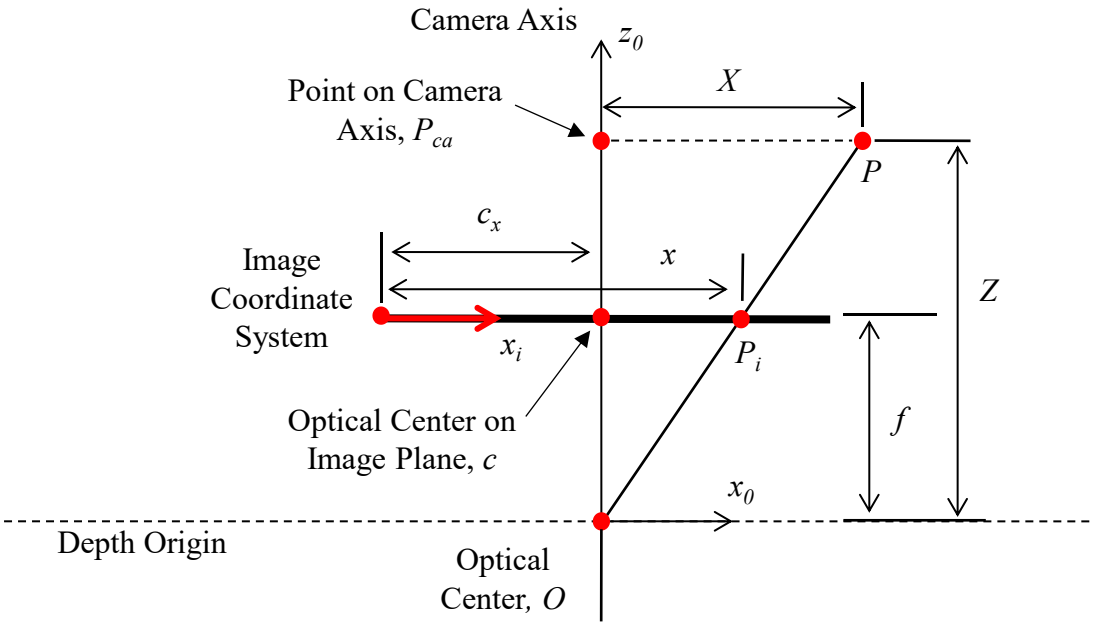


Figure 5.2. Schematic of deprojection geometry

Deprojection is derived from the pinhole model, as seen in Figure 5.2, using similar triangles. A triangle is formed from points P, O, and P_{ca}, and the second one is from points O, c, and P_i [89], [111]. The depth, Z, and pixel coordinate, x, is taken from the depth map, while the optical center pixel coordinate, c_x, is from calibration.

$$X = \frac{x - c_x}{f_x} Z \tag{5-1}$$

The x coordinate, X, of the point, P is solved from the similar triangles for (5-1).

$$Y = \frac{y - c_y}{f_y} Z \tag{5-2}$$

The same approach used for (5-1), can be used to derive the y coordinate of the 3D point as shown in (5-2). Equations (5-1) and (5-2) are derived from an ideal case, without imperfections such as radial and tangential distortions [89], [111]. The inverse Brown-Conrady distortion

model [112] can be used to correct for these errors in the pixel coordinates, x and y, before using (5-1) and (5-2).

$$r^2 = x_d^2 + y_d^2 \tag{5-3}$$

$$n = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \tag{5-4}$$

Radial distortion is modelled using (5-3) to model the radial distance, r, using the distorted pixel coordinates, x_d and y_d, and (5-4) as the high order distortion model with three distortion coefficients, k_1, k_2, and k_3 [112].

$$x_u = x_d n + 2p_1 x_d y_d + p_2(r^2 + 2x_d^2) \tag{5-5}$$

$$y_u = y_d n + 2p_2 x_d y_d + p_1(r^2 + 2y_d^2) \tag{5-6}$$

The tangential distortion is modelled by the second and third terms in (5-5) and (3-7), with distortion coefficients p_1 and p_2. The radial distortion are accounted for in (5-5) and (3-7) with the terms, x_df and y_df, to find the undistorted coordinates, x_u and y_u [112]. However, all the distortion coefficients were set to zero because Intel found this to improve the accuracy of the Intel RealSense D435 [113].

5.3 Surface Reconstruction

The deprojected depth map provides a point cloud of 3D points along the surface of the donor lung. However, the point cloud cannot be used for measurements because it does not have surfaces. To reconstruct the surface of the donor lung, and obtain measurements, the point clouds are meshed to define linear triangular element surfaces [57].

Before surface meshing, the point clouds must be down sampled, because they are needlessly dense for this application. A standard point cloud will have over 9000 points when the depth map is at its lowest depth map resolution, 240 × 480, on the Intel RealSense D435. Therefore, the

point clouds are down sampled to about 800 points to improve the speed and reduce memory requirements for future processing steps, while smoothing the extracted surface like a low pass filter.

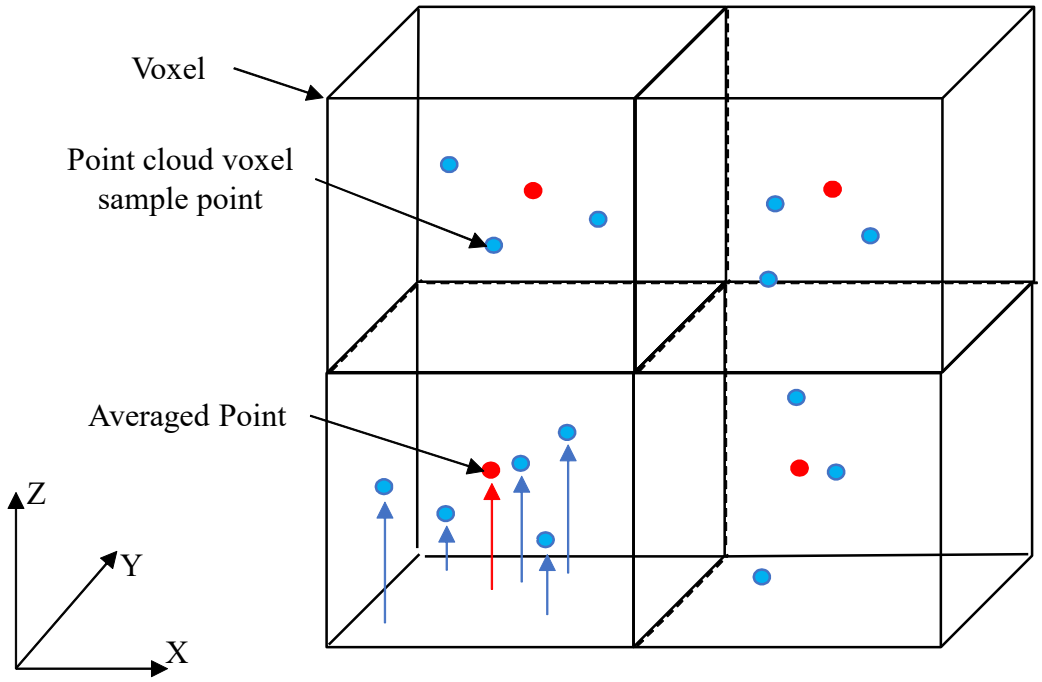


Figure 5.3. Schematic of box grid filter

The point clouds are down sampled using a box grid filter [114], as seen in Figure 5.3. The point cloud domain is segmented into 3D grids where each unit is a cube, or voxel. Each voxel returns a single averaged point with coordinates that are the average of all points inside the voxel. Box filtering has the same effects as a low pass filter, mitigating noise and smoothing the results. The effect of the box filtering is defined by the size of the voxels, density of points, and size of the domain.

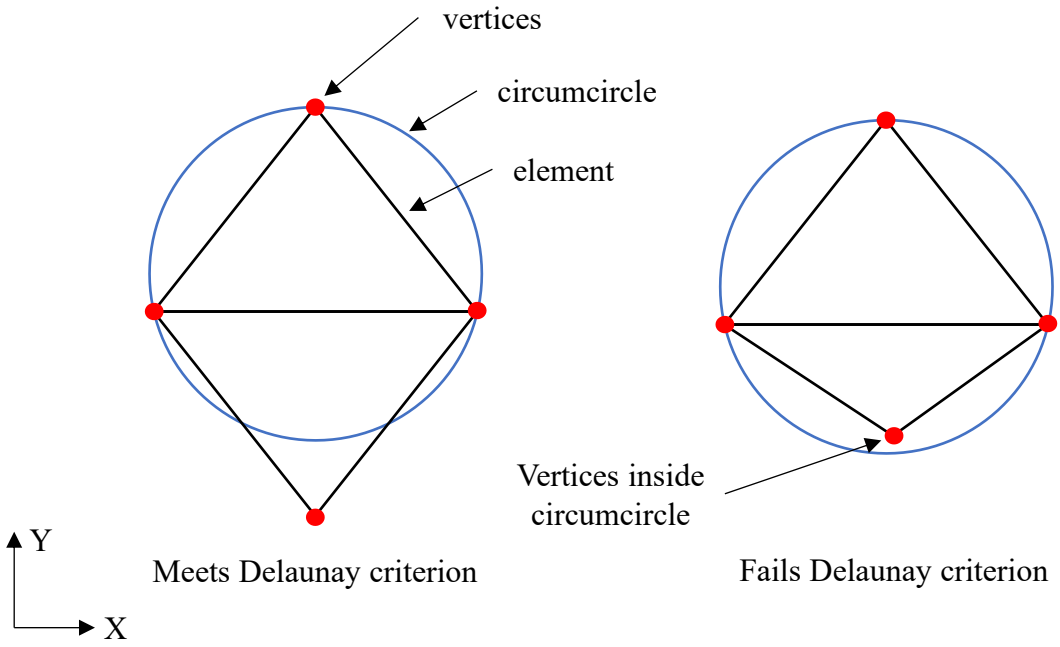


Figure 5.4. Schematic of Delaunay criterion

After down sampling, the point clouds are used to create 2D surface meshes using Delaunay triangulation. The surface mesh should be "well-shaped" because the linear triangular elements satisfy the Delaunay criterion, where no vertices are inside the circumcircle of an element, as shown in Figure 5.4 [115].

Several Delaunay triangulation algorithms exist [116], however, the MATLAB implementation is unknown because it is not stated in their public documentation. 2D surface meshes were formed using the x and y coordinates of a point cloud, then using the resultant triangulation on the 3D points.

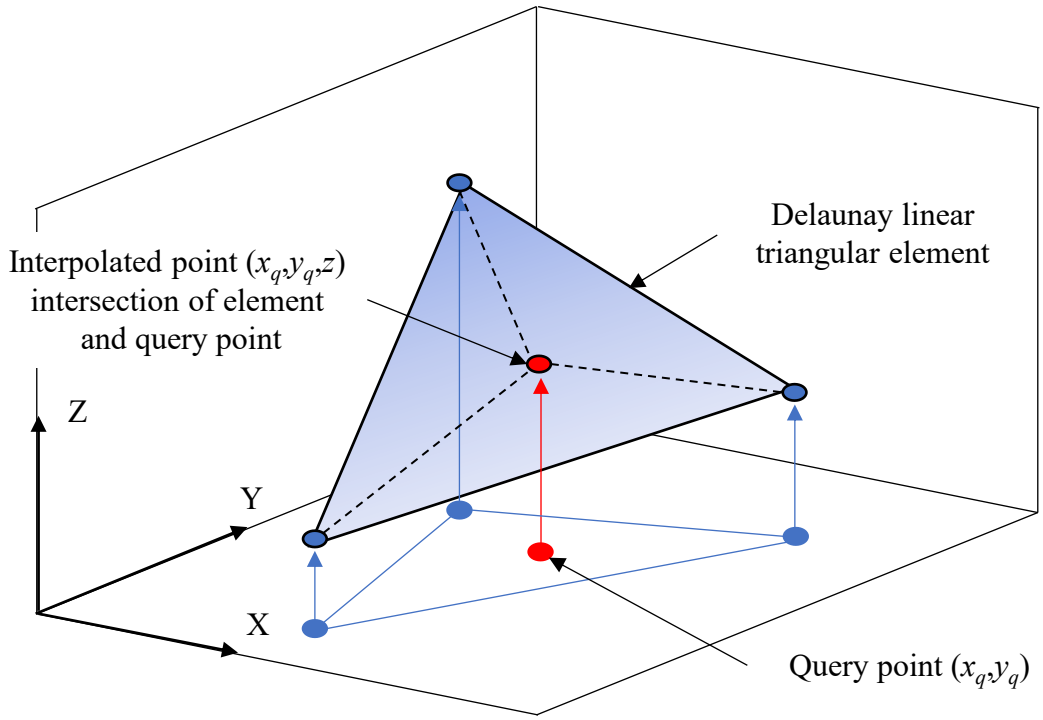


Figure 5.5. Schematic of scattered point interpolation

The point cloud data inherits limitations from active stereo vision such as not being able to track the surface of the donor lung. As a result, the number of 3D points and their x and y coordinates will be different between frames, making it difficult to compare the surface between frames. Also, some areas of the lung may be occluded creating low-resolution regions in the point cloud. To mitigate these problems, the surface mesh is refined using scattered point interpolation [117], [118]. MATLAB interpolates scattered data for height by finding the intersection of the query point and the Delaunay triangulation surface, as seen in Figure 5.5 [117]. The color and labelling data are interpolated separately using different methods but using the same neighboring points.

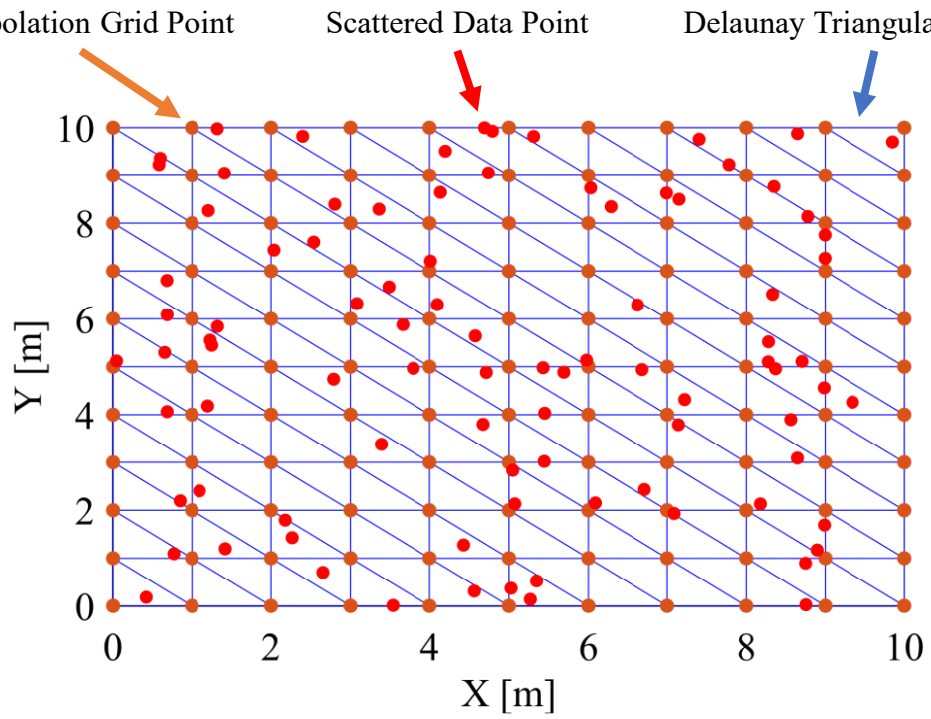


Figure 5.6. Schematic of scattered point interpolation at uniform grid points

New surfaces were interpolated from the down sampled point at query points on a uniform grid in the *XY* plane, as seen in Figure 5.6. As a result, the interpolated *z* coordinates have the same data structure as an image. This allows all interpolated images, or surface maps, to reuse of the same Delaunay triangulation. Also, their data structure allows arithmetic operations between surface maps, which allows the method to find the change in the donor lung's shape over time. Lastly, the interpolated surface maps can be enhanced using image processing methods such as a low pass filter for smoothing.

5.4 Reconstructed Surface Measurement

The reconstructed surface of the donor lung can be used to measure whole lung plethysmography metrics and to measure regional surface deformation. For instance, the lung's volume with respects to time can be estimated using the Divergence Theorem [32]. Although, this volume signal may be inaccurate, it can be used to track global surface displacement and derive standard plethysmography metrics such as tidal volume and inspiratory time. These methods provide similar measurements as the airway sensors used by EVLP devices.

Donor lung measurement can be extended by identifying areas of under and over inflation as peaks in the lung's reconstructed surface map. Also, the same approach can be used with surface maps analogous to tidal volume and dynamic compliance. Lastly, this method allows for the measurement of a typical breath for the donor lung, giving another option to track changes over EVLP treatment.

5.5 Surface Integration

Whole lung plethysmography metrics can be derived from the estimated volume of the donor lung using the reconstructed surface. De Boer *et al* proposed the Divergence Theorem to estimate the lung volume of patients by integrating the volume between the patient's anterior and posterior chest walls [32].

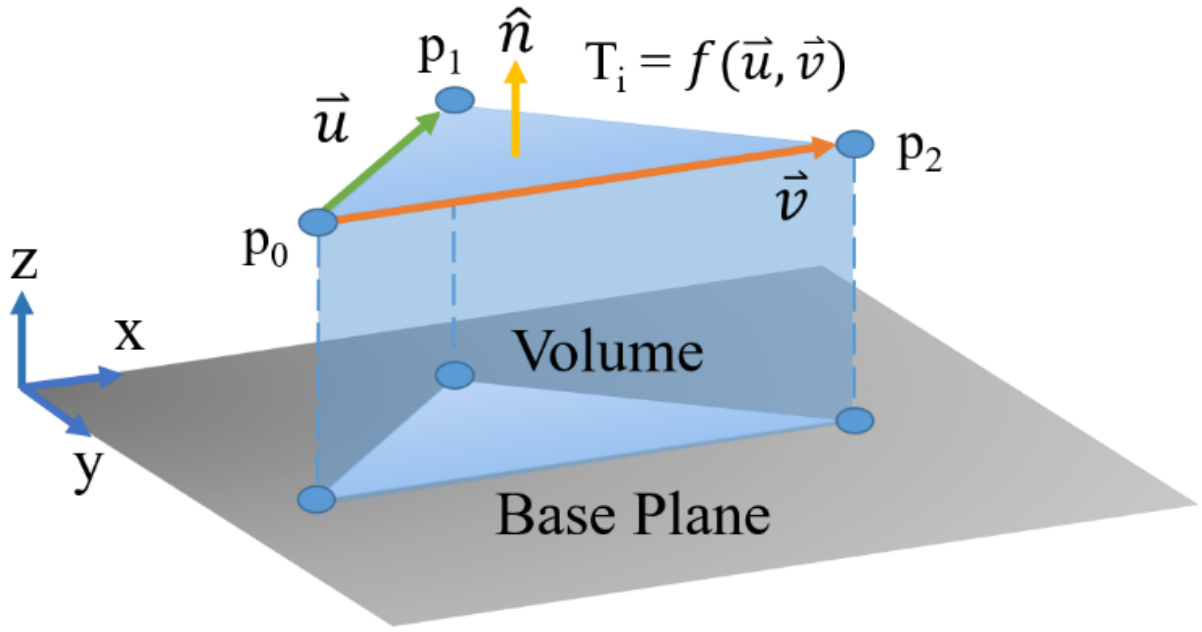


Figure 5.7. Schematic of parameterized triangular surface mesh for volume integration using the Divergence Theorem.

This problem can be modelled with two triangular surfaces, which is like the mesh elements of the reconstructed surface, as seen in Figure 5.7. De Boer et al. assumed that the posterior chest wall was flat, modelled by a plane, because they measured patients with their backs against a wall. The bottom element is assumed to be on the XY plane, or base plane in Figure 5.7. This implies its three points have the same x and y coordinates as the points directly above them, p_0,

p_1 and p_2, that define the top element, T_i. The surface vectors u and v are found as the difference between points, p_1-p_0 and p_2-p_0. The element's normal vector, $\hat{n}$, is the cross product of these two parameters. Volume integration is performed for each triangular element of the surface mesh, and the sum is taken as the lung volume.

$$\int_T (\vec{F} \cdot \hat{n}) dS = (\vec{u}\ x\ \vec{v}) \cdot \left[\left(\frac{p_0}{2} + \frac{\vec{u} + \vec{v}}{6} \right) \cdot \hat{k} \right] \hat{k} \qquad (5\text{-}7)$$

De Boer *et al* [32] proposed the closed-form solution to the volume integration of this problem (5-7), where k is the basis vector for z cartesian coordinates.

$$V = \frac{1}{6} \sum_{i=1}^{N_T} [(\overrightarrow{u_i}\ x\ \overrightarrow{v_i}) \cdot (T_{i0z} + T_{i1z} + T_{i2z})] \qquad (5\text{-}8)$$

Equation (3-7) is the discretized form of (5-7) implemented to calculate total lung volume, V, where T_{i0z}, T_{i1z}, and T_{i2z} are the z coordinates of points p_0, p_1, and p_2 that form the mesh element T_i, and N_T is the number of elements.

The volume signal is not expected to be equal to the total lung volume because, in this application, the method has several limitations. Only the top surface of the donor lung is reconstructed because it is the only surface visible to the stereo vision system. Therefore, the method ignores the side and bottom surfaces of the lung. Donor lungs are expected to have a complex and asymmetric shape that requires accurate reconstruction to obtain a reasonable volume estimate. Also, any lung movement will change the volume signal, including any horizontal and vertical displacement not caused by inflation. Lasty, the integration base plane is assumed to be static, which is likely not true because the donor lungs are suspended on a compliant surface. Therefore, the volume signal should be referred to instead as a scalar or global displacement signal of the lung since it tracks the displacement of the lung.

The displacement signal is expected to track the changes in the donor lung with respects to time and have a proportional change in magnitude with the actual lung volume. Therefore, the displacement signal was treated as a volume signal for plethysmography measurements.

The displacement signal was filtered to improve the signal quality. A low pass filter [119] was used to remove noise above the frequency of the breathing rate of the lung. Also, outliers were removed using the Hampel filter [120], followed by Savitzky-Golay [121] filtering to smooth the signal.

The derivative of the displacement, or displacement rate, found from this method, was found to be an estimate of the airway flowrate when normalized [35]. A differentiator filter was selected for this task to introduce minimal noise [122]. The flowrate signal was processed with the same filters as the volume signal. The displacement and displacement rate signals were used to measure plethysmography metrics described in Chapter 2.

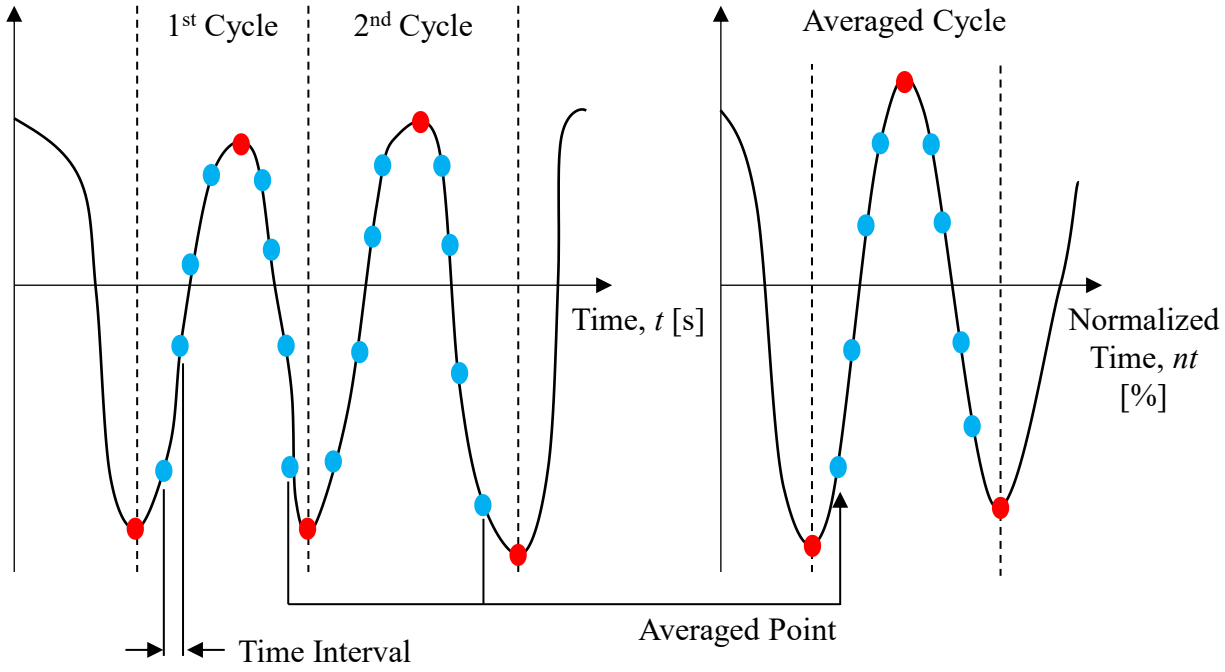


Figure 5.8. Schematic of respiratory cycle average volume

The performance of the donor lung can be characterized as an average respiratory cycle found from the displacement signal. The average respiratory cycle is found by sampling the displacement of each cycle at normalized points in time, as seen in Figure 5.8. Average cycles were found using *create_avg_cycle()* in Appendix B.

Also, the average respiratory cycle can be used to measure any changes in the donor lung's cycle by finding the difference between the sampled points and the average cycle. Furthermore, measuring this deviation from the average respiratory cycle can be used to track the long-term performance of the donor lung, as an alternative to tidal volume and dynamic compliance. This deviation was found using *compare_to_avg_cycle()* in Appendix B.

The surface map can be used to measure plethysmography, as described above, and changes across the surface of the lung by taking advantage of its data structure.

5.6 Surface Deformation and Regional Measurement

The surface map can be used to measure regional performance of the donor lung and identify localized over-inflation. Its data structure is like an image, with an intensity value at a point on a uniform grid, or matrix. This coincidence allows the surface map to be enhanced using digital image processing methods such as digital filters for smoothing [123] and noise removal. Also, other algorithms could be used for pre-processing such as non-maximal suppression from Canny edge detection. These processes prepare the surface map for peak detection and regional displacement measurement.

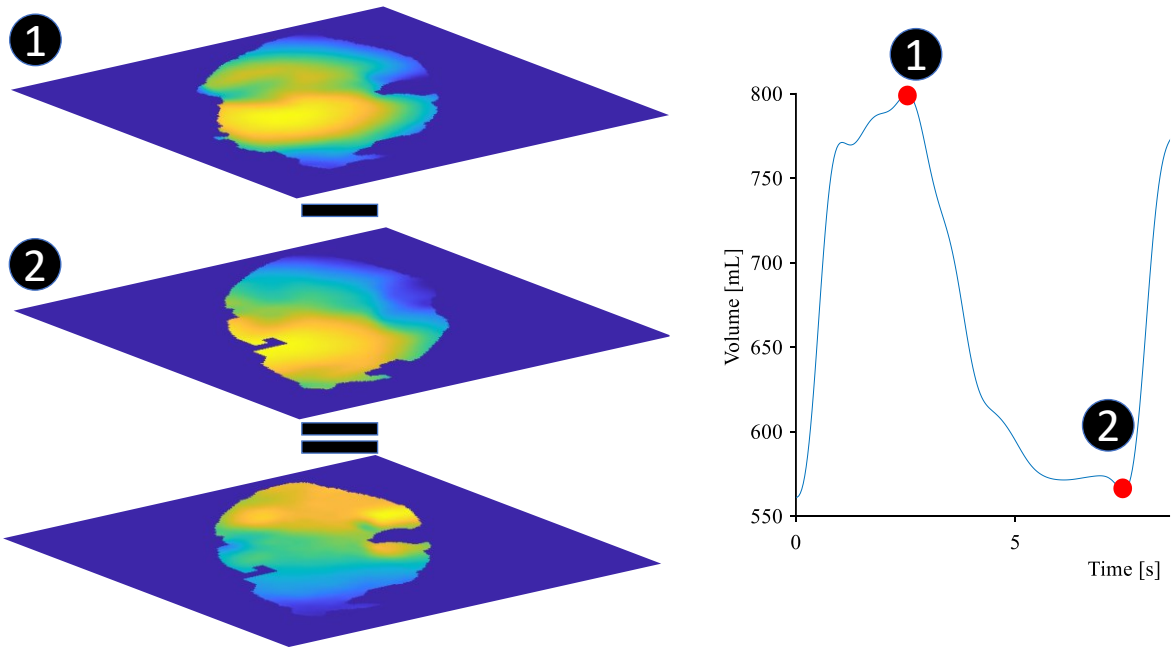


Figure 5.9. Schematic for tidal displacement map calculation

Regional measurements can be obtained from the surface maps through arithmetic operations between images. For example, the pixel-wise difference of two surface maps measures the regional displacement in that time frame, treated as an estimate of regional inflation. Therefore, the difference of surface maps taken at the start of inhale and exhale gives regional inflation akin to tidal volume, as seen in Figure 5.9. This surface tidal displacement can be used to identify where and how much the donor lung distends. Also, surface dynamic compliance can be found by scaling the surface tidal displacement by the peak pressure change in the respiratory cycle, just like scalar tidal volume.

The surface maps can be used to characterize the donor lung by its average surface shape during a respiratory cycle. This method is like the average respiratory cycle displacement described in Chapter 5.5 except applied to all the query points of the surface map. Any differences from this average respiratory cycle surface map would indicate changes to the donor lung's performance. The respiratory cyclic average could be calculated by sampling the height of each point in the reconstructed surface at normalized time intervals within each respiratory cycle. These samples

would be averaged at each point, and each point in the respiratory cycle to get the averaged surface.

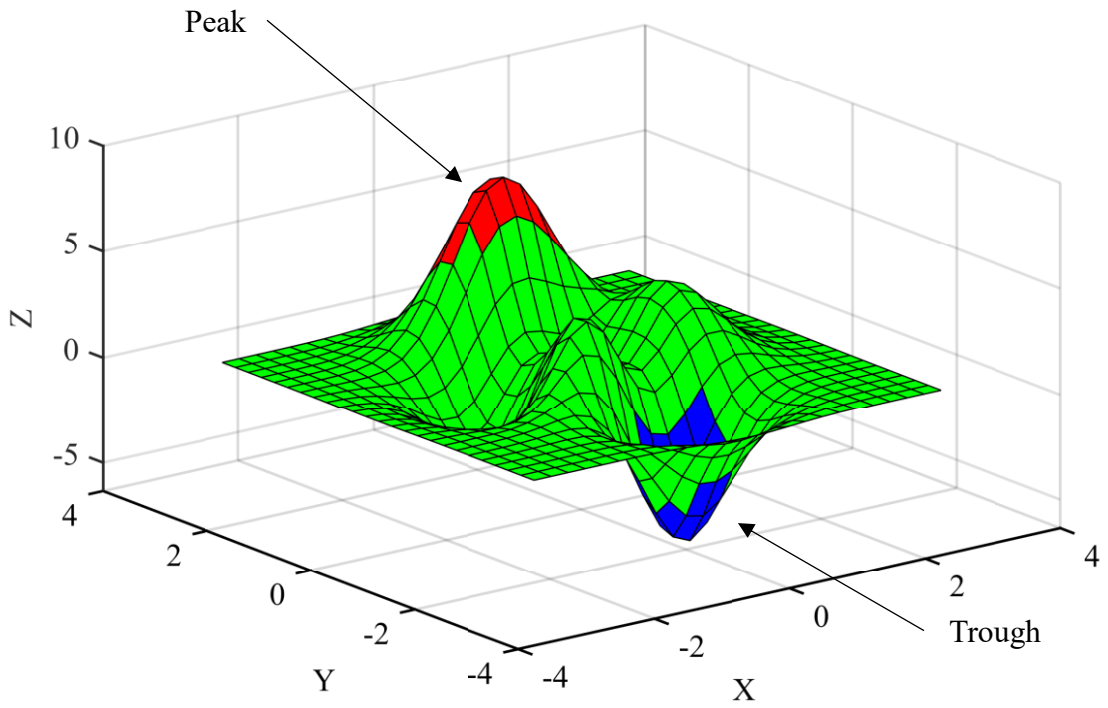


Figure 5.10. Schematic of peak detection of a surface annotated for peak and trough detection

In an ideal case, surface tidal displacement will be a flat surface, indicating the donor lung has homogeneous performance. However, if regions with different displacement form, then this indicates the donor lung has heterogenic performance. If a peak in performance forms, then it may indicate localized over-inflation, or the onset of ventilation induced lung injury (VILI). Also, these regions will exist in the surface dynamic compliance. These regions, peaks, and troughs, can be detected using image processing methods, as seen in Figure 5.10.

Peaks can be detected in the regional measurements using digital image processing techniques including the image segmentation described in Chapter 5.1 . For example, segmented regions with a high average intensity could be identified as peak regions. Another example would be local maxima [124] and minima [125] detection. The simplest method would be thresholding height, or the surface map intensity. These methods could be categorized into global and local methods.

The surface map takes on the shape of the donor lung, therefore, will have local and global variation that would be difficult to account for when globally searching for peaks. Therefore, the surface map should be searched using a local method such as an area-based digital filter for outlier detection. The surface tidal displacement and dynamic compliance are suitable for both local and global methods since they are ideally flat with no local variation. Lastly, region and peak detection could be performed on other surface metrics such as mesh curvature.

5.7 Processing Scheme Conclusion

Table 5.1 Measurement Method Feature Comparison

Measurements	EVLP	ASV
Airway Flow Rate	✓	×
Airway Pressure	✓	×
Tidal Volume	✓	✓
Displacement Signal	×	✓
Displacement Rate	×	✓
Surface Shape	×	✓
Surface Tidal Displacement	×	✓
Surface Displacement Cycle Average	×	✓
Surface Local Minima and Maxima Detection	×	✓

A processing scheme was developed to measure global and regional performance of a donor lung using depth maps from the Intel RealSense D435. The method's measurements are contrasted with the key EVLP measurements in Table 5.1. The depth maps were segmented to select depth measurements of the lung's surface. These measurements were deprojected into 3D points along the surface of the lung, called a point cloud. The lung's surface was reconstructed by performing scattered point interpolation and Delaunay triangulation on the point clouds. The reconstructed surface, or surface maps, was integrated to estimate lung volume and derive plethysmography metrics such as tidal volume and dynamic compliance. The volume signal is not expected to be an accurate estimate of total lung volume due to method limitations. However, it is expected to change with respects to lung inflation and displacement in time, allowing it to be treated like volume for plethysmography. Changes in the surface maps with time are measured by finding the difference between frames to obtain surface metrics that are like tidal volume and dynamic compliance. Also, the surface maps can be scanned to identify peaks that may be localized over-inflation and the onset of VILI.

6 Validation of Displacement Measurement

A ventilator test lung was measured to evaluate the active stereo vision plethysmography system by comparing its measurements with an EVLP respirometer system. The ventilator test lung was used as a testing surrogate for donor lungs because it distends with ventilation and is designed to be mechanically ventilated. Measurements were taken during positive pressure mechanical ventilation, performed by the EVLP. Global and regional plethysmography measurements were derived from the depth maps from the Intel RealSense D435. However, only the scalar measurements, such as tidal volume, were compared to the EVLP to measure correlation and agreement. If these measurements have a high correlation and agreement, it may suggest the regional measurements are valid as well.

6.1 Experiment Equipment

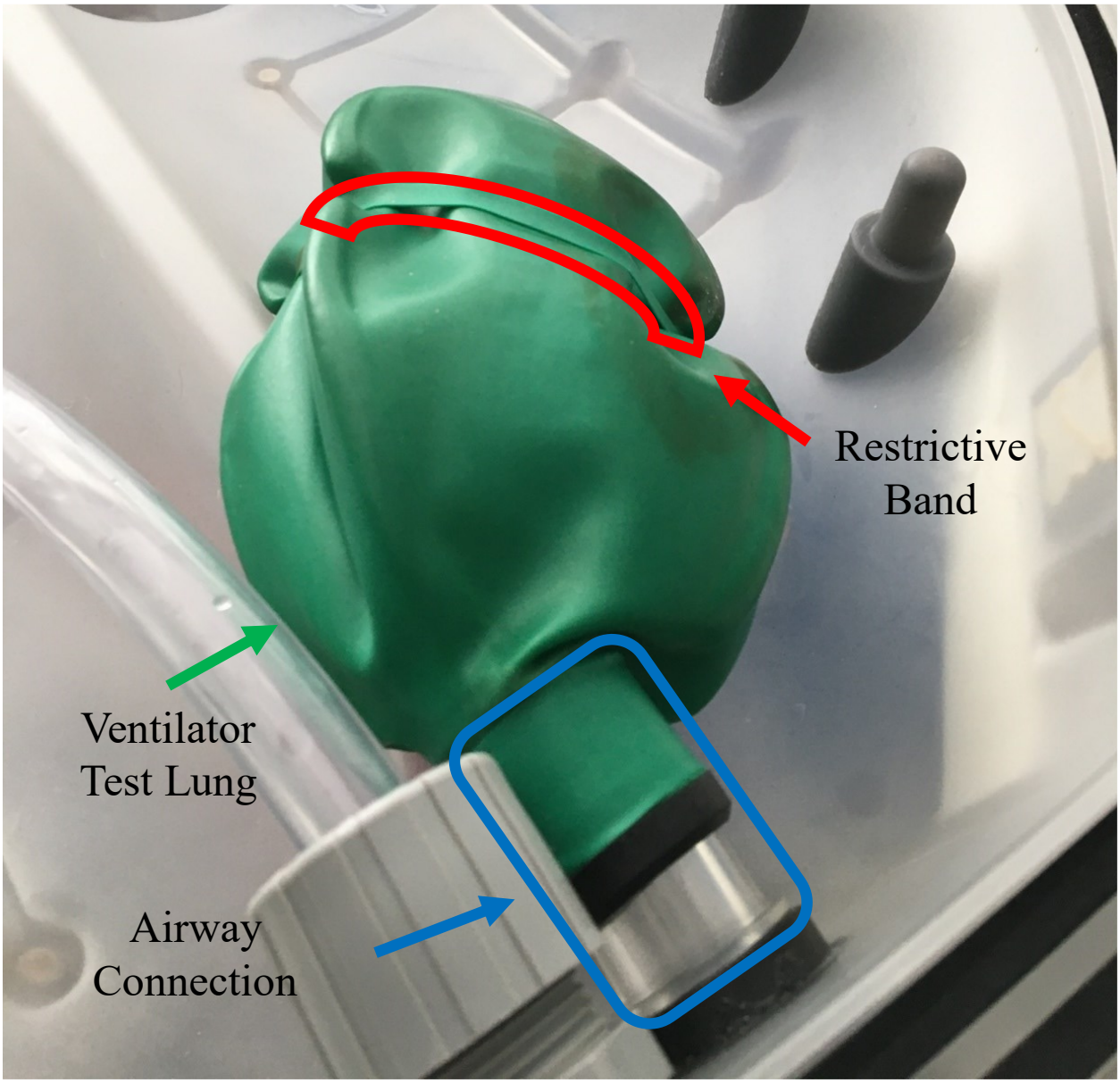


Figure 6.1 Annotated image of the ventilator test lung

The ventilator test lung, shown in Figure 6.1, is a latex balloon filled with a dense foam intended to test the performance of ventilators. Its capacity is approximately one liter, and it comes with a standard 22mm ID connector pressure port to connect with ventilators.

The ventilator test lung was selected for this experiment because it is an acceptable substitute for a real lung because it deforms with inflation and has a complex surface shape. Also, the test lung

is convenient to operate with a ventilator, as it is intended to measure ventilator performance. Notably, it is flat when fully collapsed, and has a restrictive band around its mid-section creating two lobes. The test lung was ventilated using Tevosol's development EVLP device for the study.

The development EVLP is designed to perform negative pressure ventilation. This is achieved by connecting the lung to a flow loop inside a sealed pressure chamber. The EVLP is operated using custom software, which receives feedback from a combined volume flow rate and pressure sensor imbedded in the flow loop. This device was selected for this study because its velocity and pressure measurements can be used to calculate plethysmography metrics that can validate the active stereo vision method measurements.

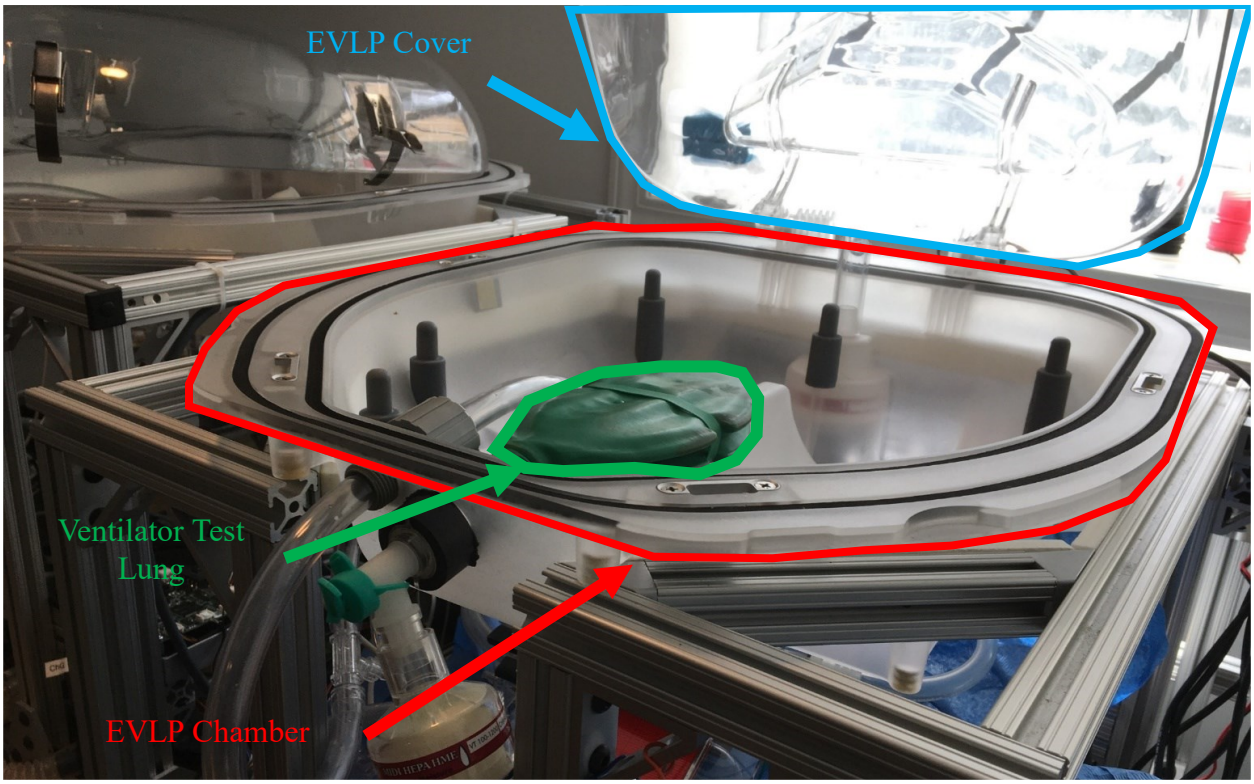


Figure 6.2 Annotated image of the ventilation test lung inside a development EVLP

As shown in Figure 6.2, the test lung was directly connected to the EVLP airway through a standard pressure port, resting inside the EVLP's chamber. To measure the ventilator test lung's surface displacment with the active stereo vision system, the EVLP's chamber cover was lifted. As a result, negative pressure ventilation could not be performed. Instead, the ventilator test lung was positive pressure ventilated by the EVLP by modifying its operating procedure to perform constant positive airway pressure ventilation.

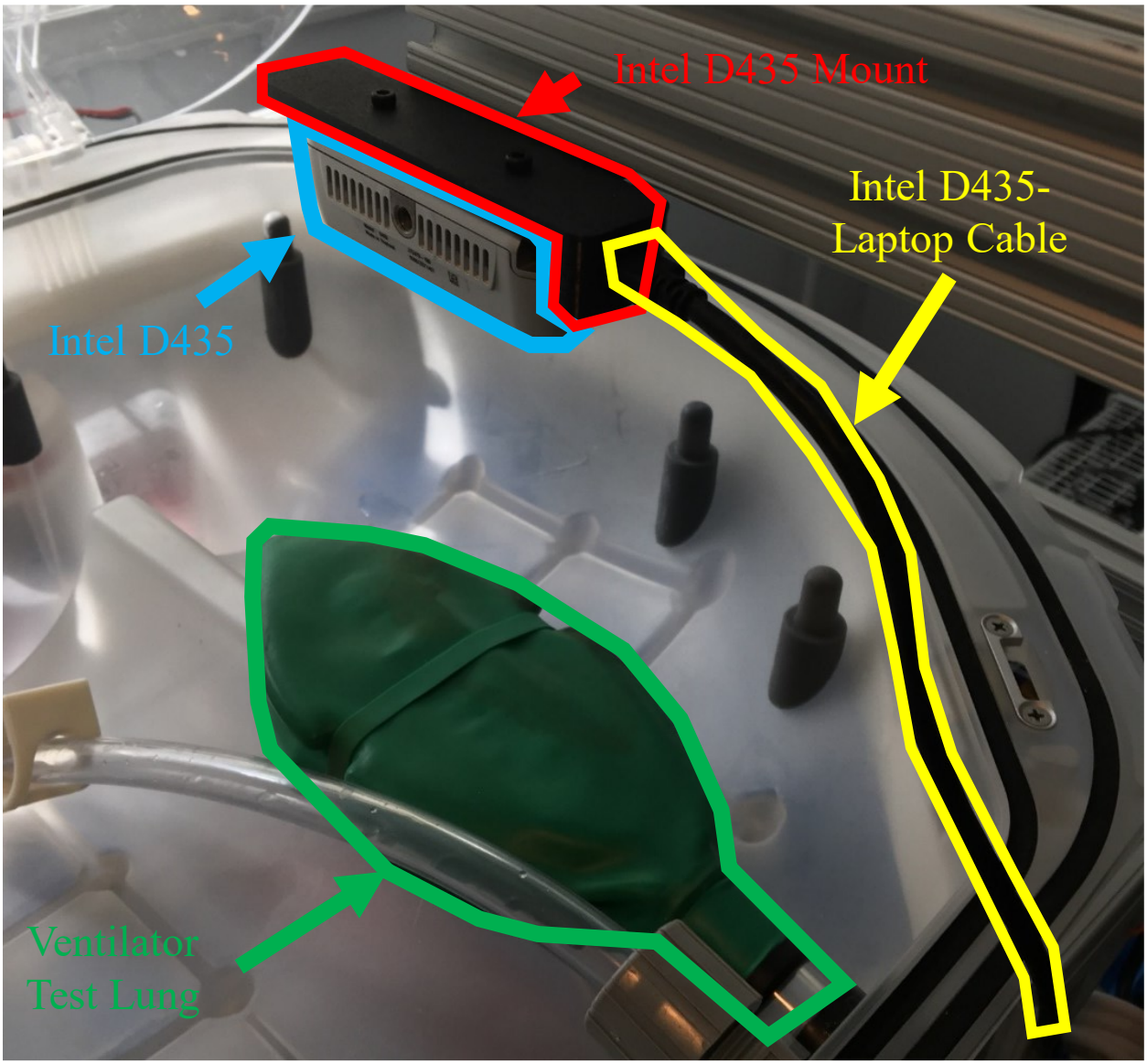


Figure 6.3 Annotated image of the Intel RealSense D435 mounted above the ventilator test lung

The Intel RealSense D435 was mounted facing downward directly above the ventilator test lung using a tripod, horizontal beam, and a custom 3D printed mount, as shown in Figure 6.3. Also, this mount supported the USB cable that connects the Intel RealSense D435 to a laptop that was used to operate the system and store measurements. Notably, the horizontal beam was leveled prior to calibration and data acquisition.

6.2 Calibration and Data Acquisition

The ventilator test lung was measured by the EVLP and Intel RealSense D435, during ventilation. The EVLP measured flow rate and pressure, and the Intel RealSense D435 performed active stereo vision to estimate the distance of the lung's surface, providing its shape.

6.2.1 Experiment Cases

Measurements were taken in multiple experiment cases to measure performance in a range of conditions. Tidal volume was varied between experiment cases, since the Intel RealSense D435 can only measure surface changes, such as inflation, of the ventilator test lung. Tidal volume was controlled indirectly by changing the EVLP blower rate setting, which controls the flow rate during the inspiratory phase of each breath. A tidal volume setting does not exist on the EVLP because it was modified to perform positive pressure ventilation from negative pressure ventilation with constant positive airway pressure. Also, camera height was varied to evaluate the effect of spatial resolution on the final measurements. The height was manually adjusted using a tripod.

Table 6.1 Ventilation test lung experiment cases

Experiment Case	ASV Height [mm]	Airway Flow Rate [%]
1	200	20
2	200	24
3	200	28
4	330	20
5	330	24
6	330	28
7	370	20
8	370	24
9	370	28

The nine experiment cases, as seen in Table 6.1, are combinations of three levels of camera height and blower rate. The blower rate is measured as a percentage of the maximum possible flow rate. The camera height was measured using a ruler between the EVLP chamber and Intel RealSense D435 depth origin. Other settings remained constant between experiment cases, tabulated in Appendix C, so they do not affect the measurement comparison.

6.2.2 **Description of the EVLP**

The EVLP uses a flow sensor to monitor tidal volume and dynamic compliance. The sensor is a Siargo FS6122 that measures flow rate, pressure, temperature, and humidity using integrated MEMS sensors. The data sheet reports the total error band in percentage of full scale (FS) for flow rate ± (2.5 + 0.5 FS) % and pressure ± 1.0 % FS. Also, the sensor has a response time of 1.8 ms [126].

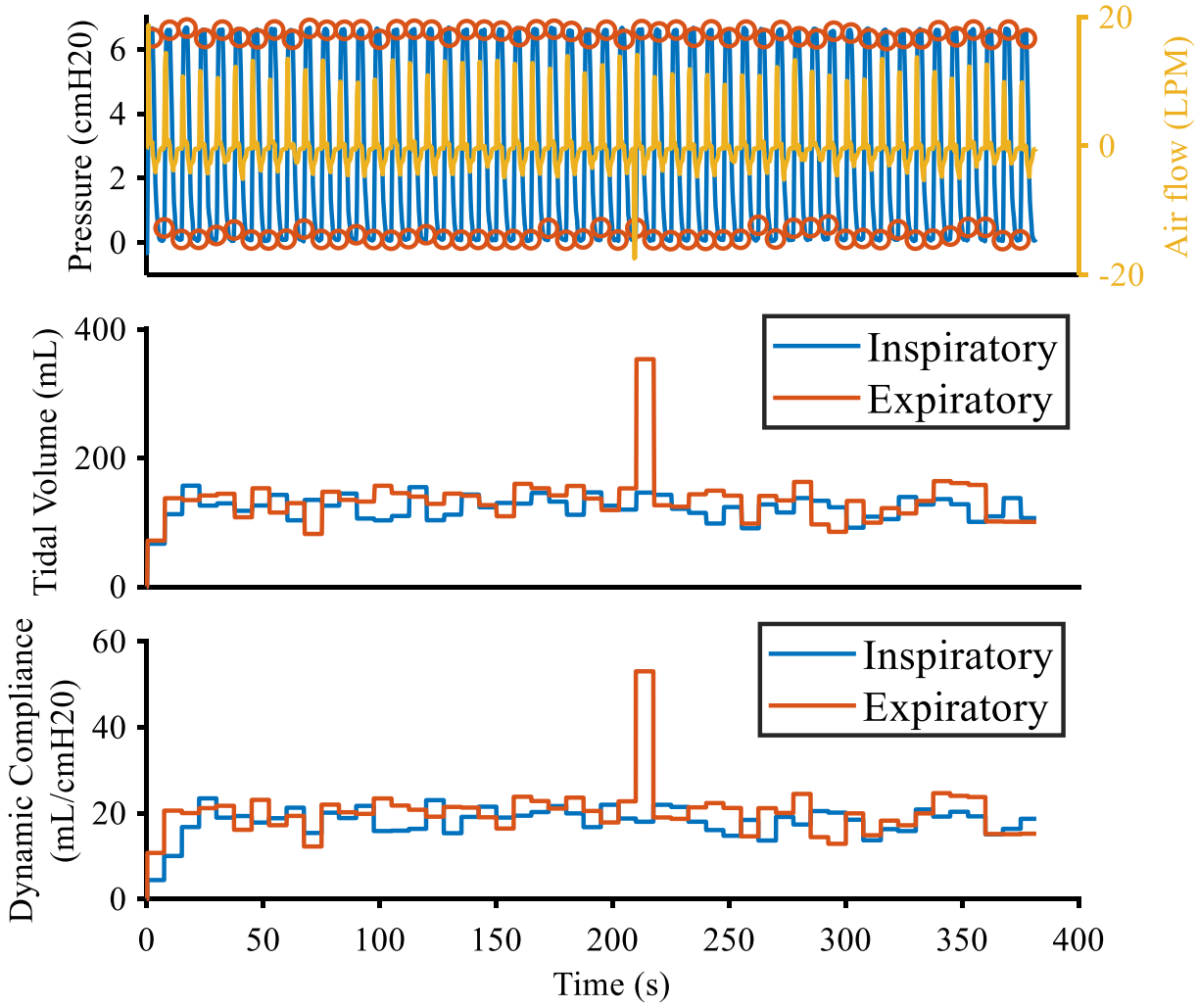


Figure 6.4 Plots of EVLP flow rate, pressure, tidal volume, and dynamic compliance measurements

Both the airway flow rate and pressure were recorded during all experiment cases, as seen in Figure 6.4. They were used to calculate plethysmography measurements such as tidal volume and dynamic compliance.

Tidal volume was calculated by integrating the airway flow rate measurements, summing the integrated volume between the start and end of each cycle. The dynamic compliance was calculated by dividing the tidal volume by the *PIP* and *PEEP* of each breath, using (3-7). The *PIP* and *PEEP* were identified in the pressure measurements based on their definitions. Other plethysmography measurements, such as inspiratory time and respiratory, were calculated based on their definitions discussed in Chapter 5. Notably, the calculations were performed by the EVLP custom software, and were provided along with the airway flow rate and pressure measurements.

6.2.3 **Data Acquisition using the Intel RealSense D435**

The Intel RealSense Dynamic Calibration application with a checkerboard calibration plate was used to calibrate the extrinsic parameters of the Intel RealSense D435. Also, the Intel Realsense Depth Quality Tool was used to evaluate the depth measurements of the Intel RealSense D435. Both software applications were used as described in Chapter 4. The intrinsic parameters, extrinsic parameters, and depth quality metrics from calibration are listed in Appendix C.

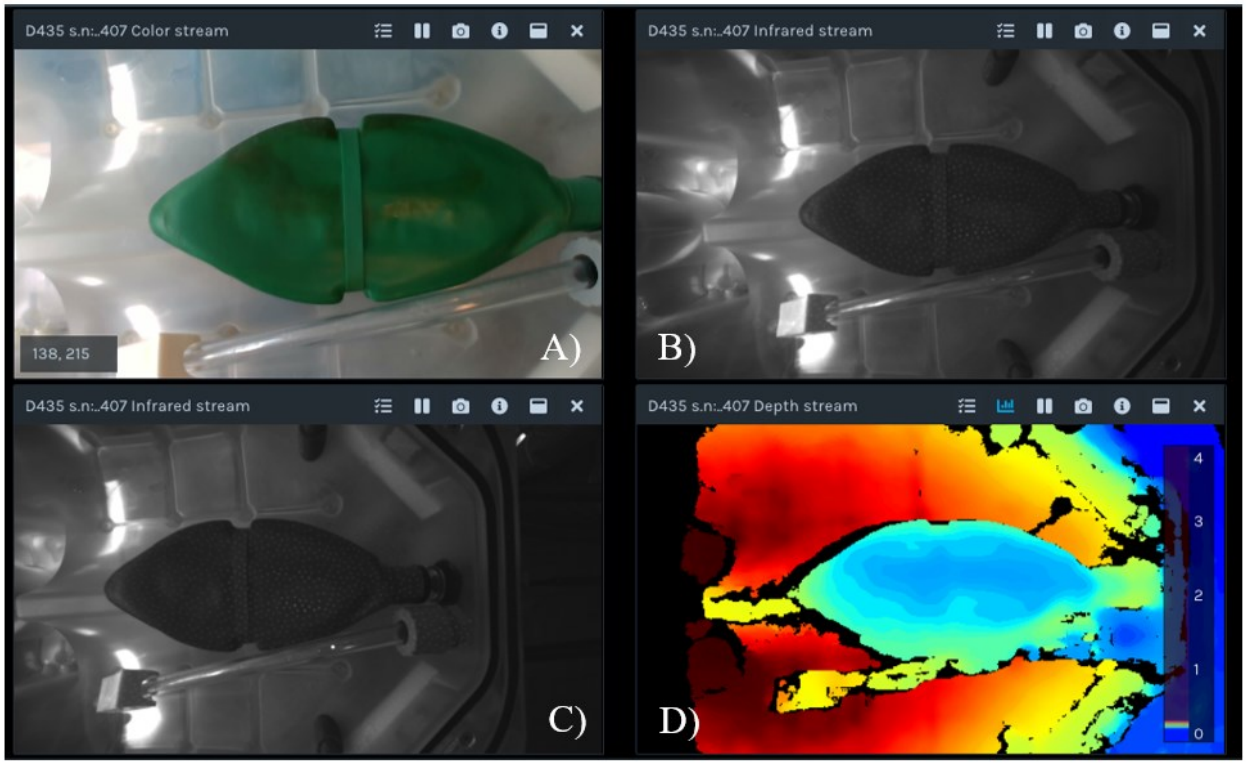


Figure 6.5 Screenshot of the ventilation test lung (a) color, (b) left and (c) right infrared, and (d) colorized depth map video stream using the Intel RealSense Viewer

The Intel RealSense D435 was operated using the Intel RealSense Viewer to record the depth map stream, as seen in Figure 6.5 (d), and metadata such as timestamps and camera intrinsic parameters. Also, the depth maps were spatially and temporally filtered using the Intel RealSense SDK. The color stream, Figure 6.5 (a), and the left and right infrared camera streams, Figure 6.5 (b) and (c), were not recorded due to memory limitations. For all experiment configurations, the depth map resolution was 480 × 848 pixels, and the sampling rate was 30 fps. Approximately 5 minutes of data was recorded for each configuration, sampling about 40 cycles, as the test lung was ventilated at a rate of 8 breaths per minute.

6.3 Processing Scheme for Active Stereo Vision Plethysmography of the Ventilator Test Lung

The experimental data obtained from the Intel RealSense D435 was processed to measure the displacement of the ventilator test lung. The processing scheme outlined in Chapter 5 was performed in MATLAB. This process begins with obtaining the depth maps from the rosbag files saved by the Intel RealSense Viewer, as described in Chapter 4.

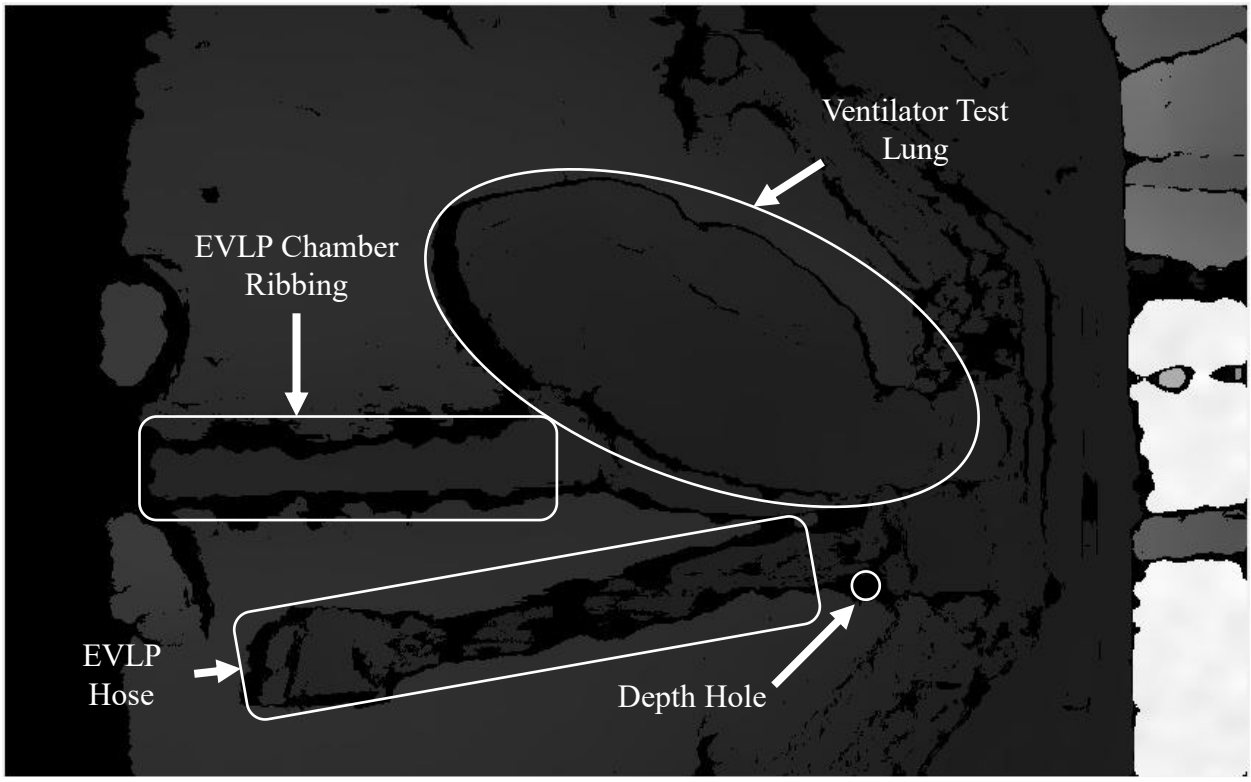


Figure 6.6 Annotated depth map, which has been spatial and temporal smoothed, of the ventilator test lung inside the EVLP chamber

The depth maps and metadata, including timestamps, were accessed in MATLAB using the Intel SDK MATLAB wrapper. When accessed, the depth maps were smoothed using the SDK temporal and spatial post-processing filters, as shown in Figure 6.6. The post-processed depth maps have depth holes, zero depth black pixels, which define borders between objects, such as the transparent hose below the test lung and EVLP chamber ribbing. These depth holes, caused by occlusion, can be utilized to segment the test lung from the image.

6.3.1 **Depth Map Segmentation**

The ventilator test lung was segmented from the depth map using a three-step edge-based segmentation method. The three main steps were: edge detection and linking, interactive segmentation followed by morphological operations, and outlier rejection with binary map temporal smoothing.

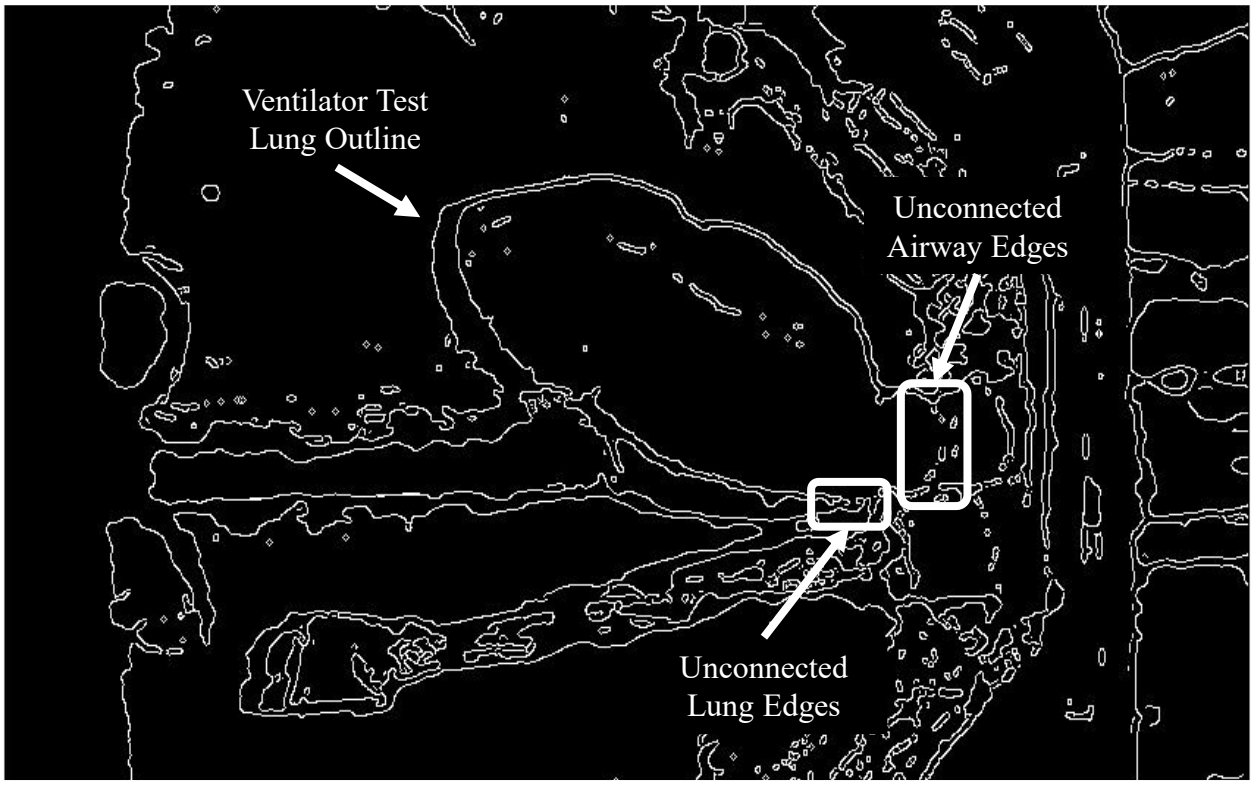
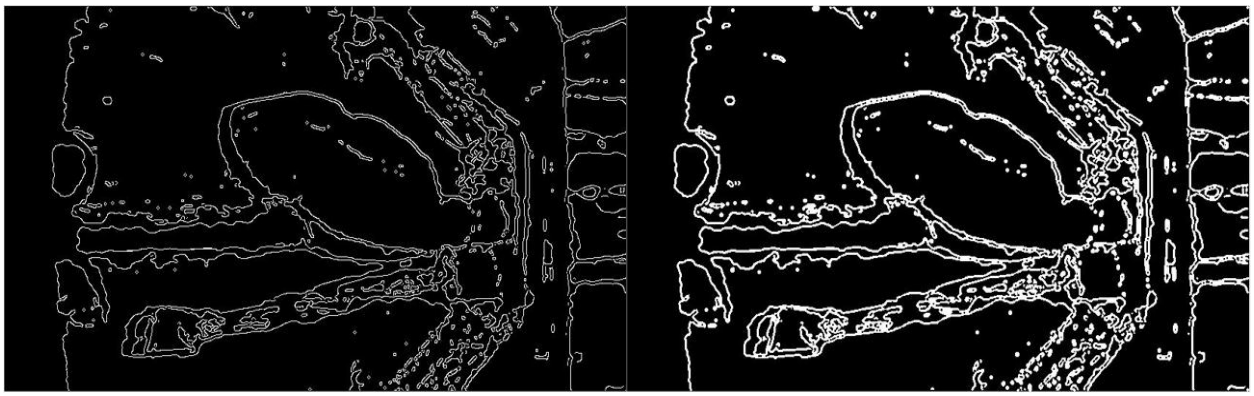


Figure 6.7 Annotated edge map of the ventilator test lung inside the EVLP chamber

The Canny edge detector was used on Figure 6.6 to obtain an edge map, as seen in Figure 6.7. Generally, the edge map are only the edges of the depth holes but do provide an outline of the ventilator test lung. However, the edges do not enclose the test lung region, as there are some unconnected edges along the bottom edge of the test lung near the airway connection. Also, the end of the test lung at the airway connection is not clearly defined. Therefore, to enclose the test lung region, edge linking was performed to connect neighboring edges. A simplified algorithm for edge linking was adapted from [100]:

1. Scan a row of the edge map for edges with a gap smaller than a threshold, fill these gaps
2. Repeat step 1 for each row in the edge map
3. Rotate the image by a desired angle, then repeat steps 1 and 2, then rotate back
4. Repeat steps 1-3 for all desired angles



(a) *(b)*

Figure 6.8 Image of the edge map of the ventilator test lung (a) before and (b) after pre-linking morphological operations

Morphological operations were used to improve the linking results before edge linking, as seen in Figure 6.8 (a) and (b). The pre-linking steps were edge thickening, diagonal fill, and the majority filter. The detected edges are mostly one pixel thin. Thickening the edges before linking allows edge linking between adjacent rows. Also, edges were linked diagonally using a morphological operation to reduce the reliance on the rotation step. Lastly, isolated points were removed using the majority morphological operation.

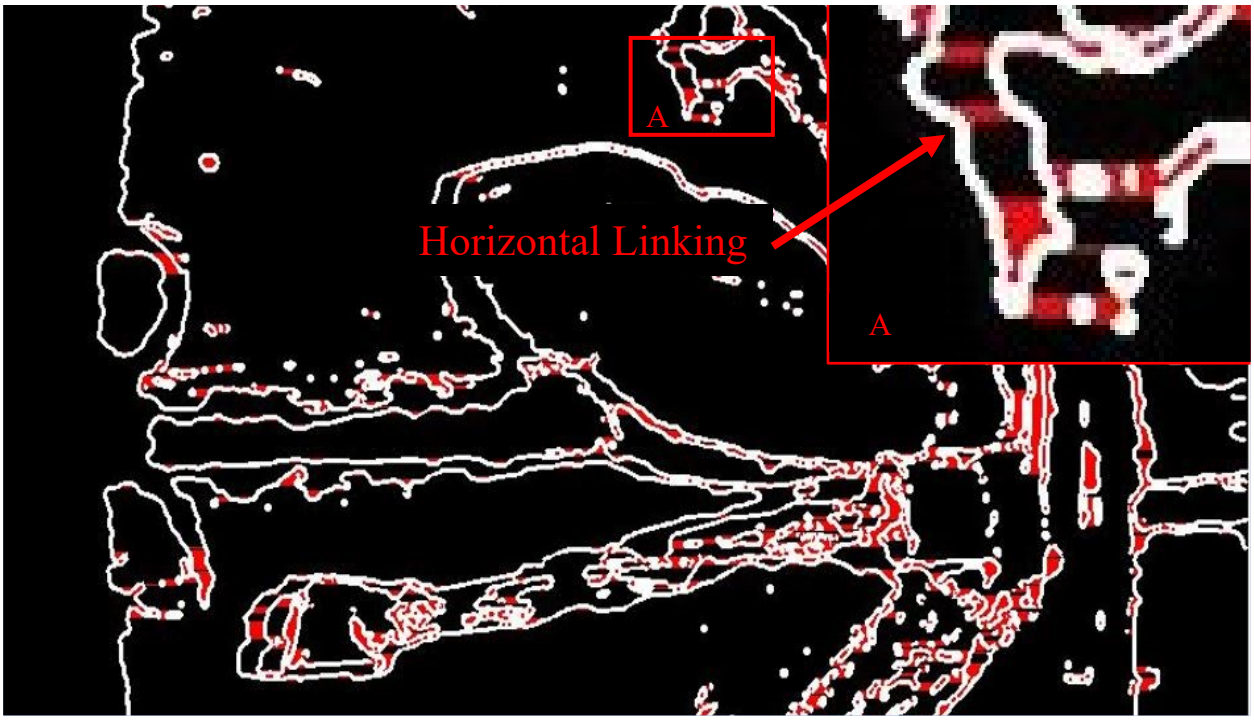


Figure 6.9 Annotated image of the edge map of the ventilator test lung after edge-linking

As mentioned, horizonal gaps are filled between edges if the gap is smaller than a threshold, as seen in Figure 6.9. This step improves segmentation by connecting the discontinuous edges along the bottom of the ventilator test lung outline. However, the first edge linking step did not fully enclose the outline because it did not vertically link the airway connection.

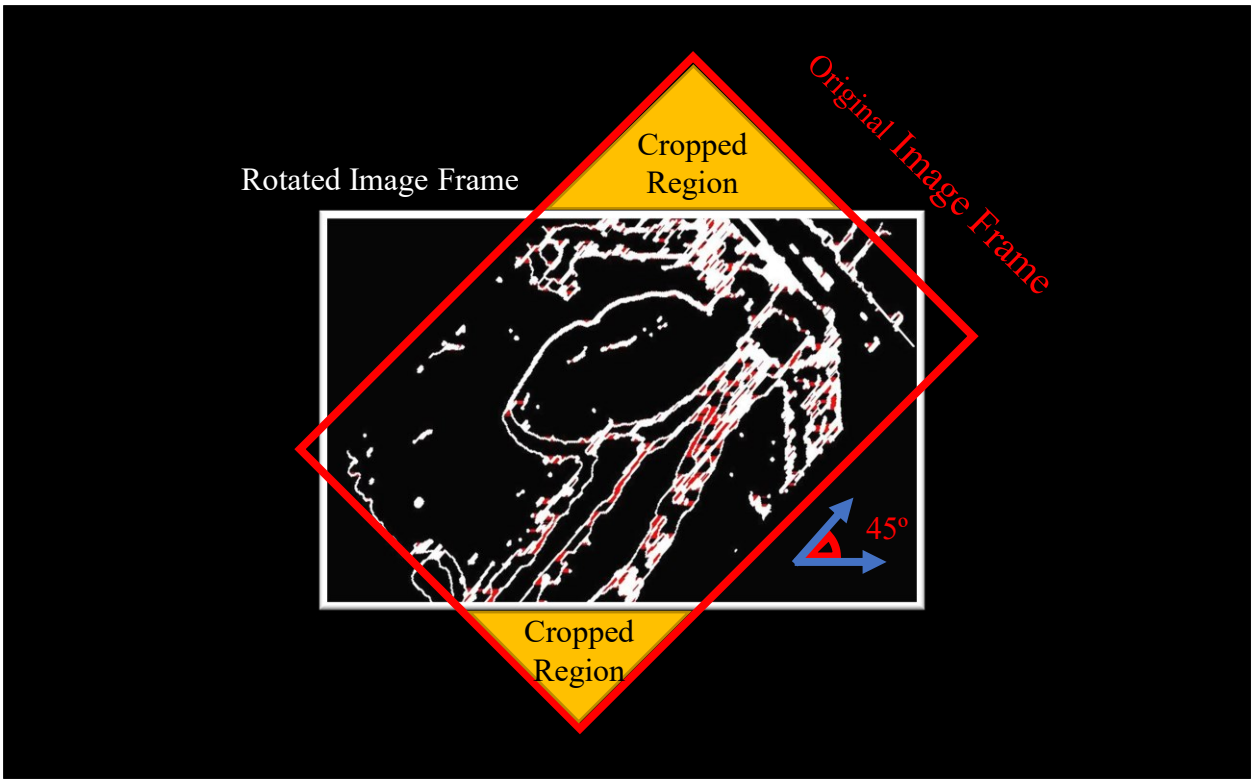


Figure 6.10 Annotated image of the edge map of the ventilator test lung after a 45° rotation and edge-linking

The edge linking algorithm was repeated three more times, each time after rotating the edge map by 45°, -45°, or 90°. Rotating the image crops the corners of the edge map, as seen in Figure 6.10. The rotated edge map is horizontally edge linked before rotating it in the opposite direction to return the edge map to its original orientation and size. This process allows rough edge linking in multiple directions.

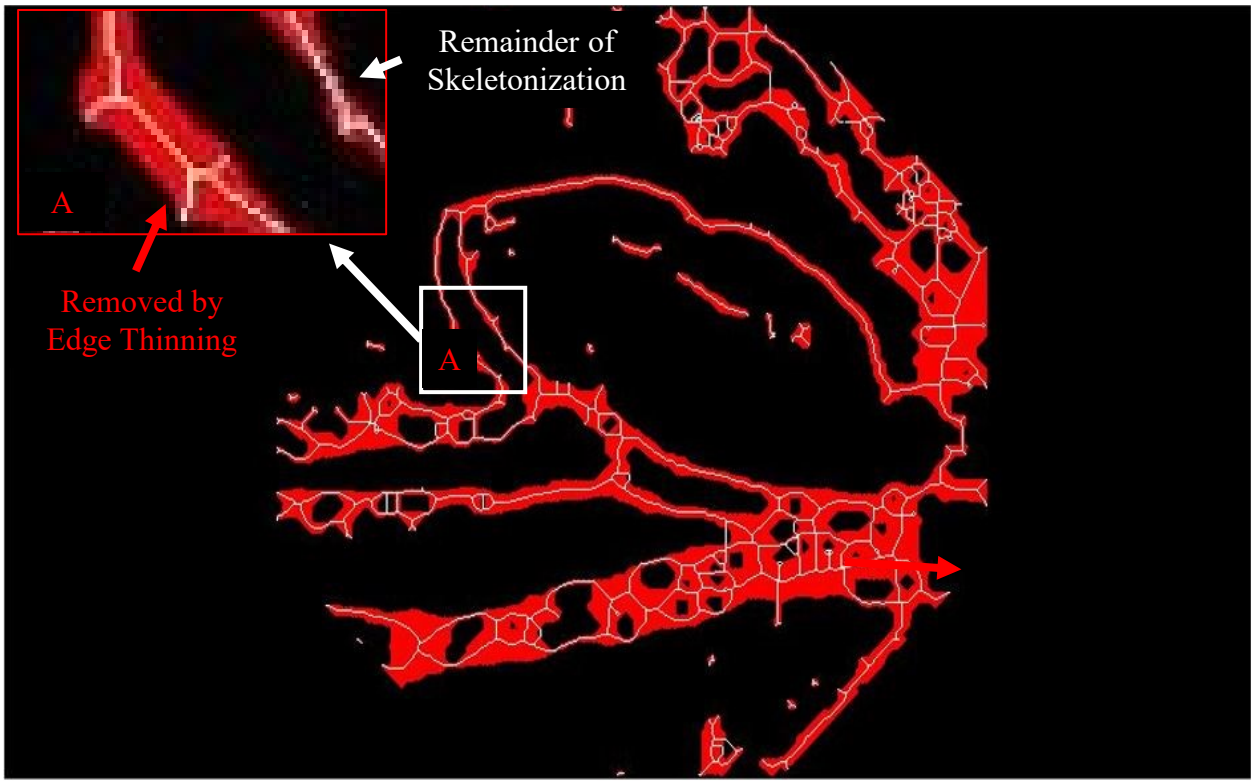


Figure 6.11 Image of the ventilator test lung edge map after edge thinning skeletonization, with removed edges in red and the remainder in white

Morphological operations were used on the edge maps after edge linking to perform edge thinning, as seen in Figure 6.11. Skeletonization removed the red pixels, leaving only the white pixels in Figure 6.11. In most edge maps, the remaining edges enclose the lung region. The edge maps were post-processed to remove artifacts such as spurs, isolated edges, and branch like structures extending from the lung outline.

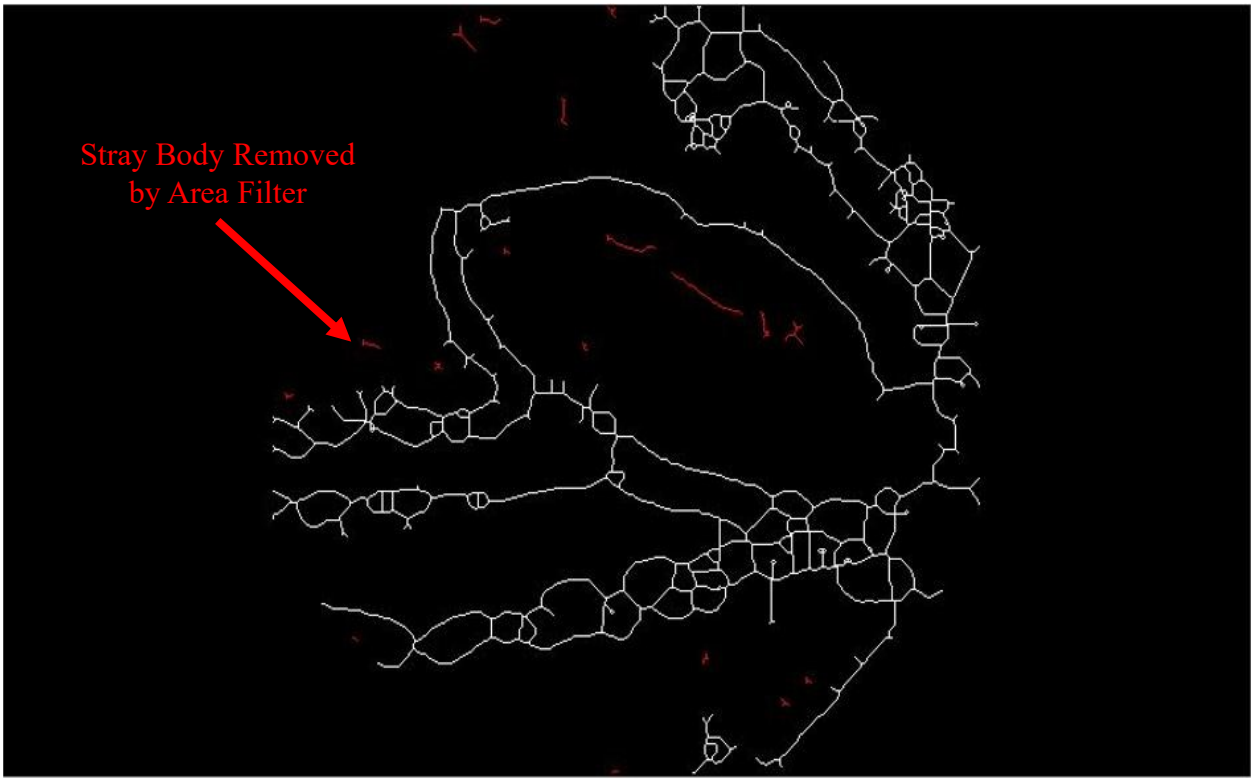


Figure 6.12 Annotated image of the ventilator test lung edge map after removing small connected components with an areal filter, with removed edges in red and the remainder in white

An areal filter removed small bodies from the edge map, leaving only the white pixels in Figure 6.12. Notably, the corners of the edge map were cropped during the edge linking rotations. However, the ventilator test lung outline was not cropped because it at the center of the edge map.

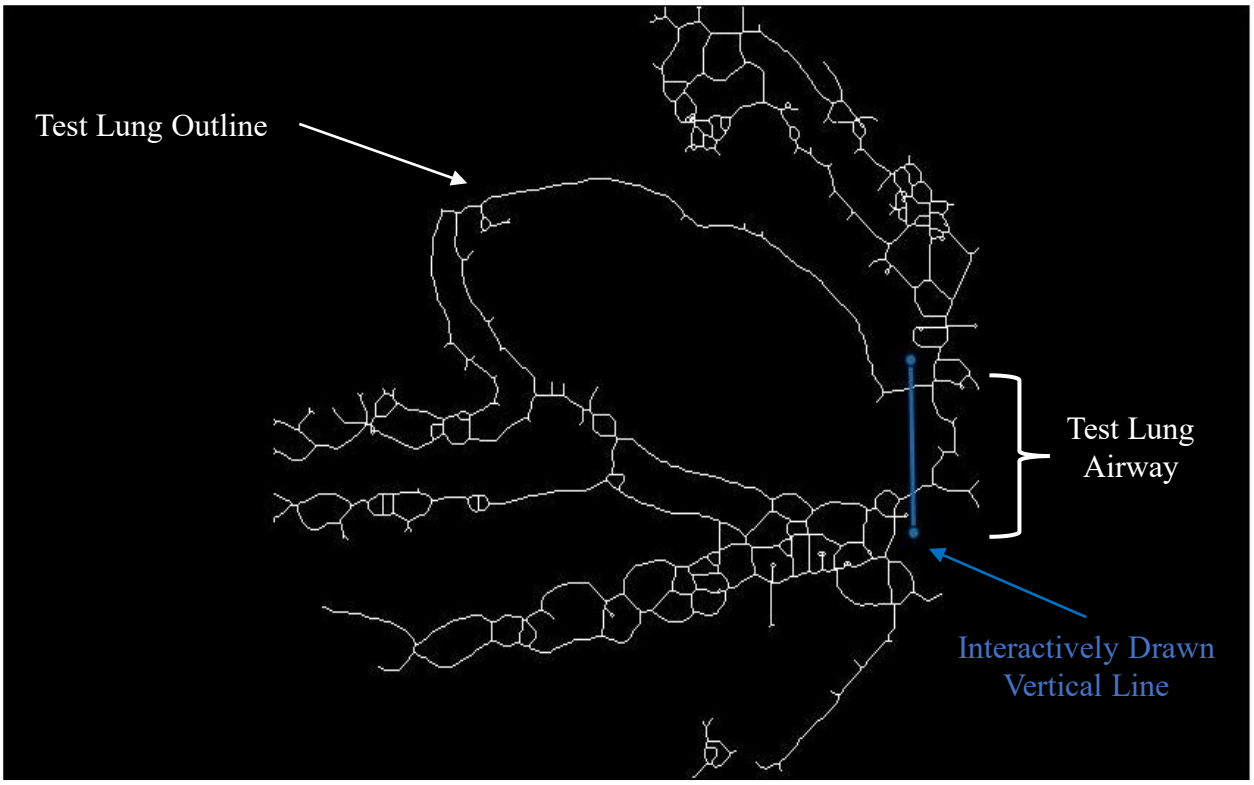


Figure 6.13 Annotated image of the ventilator test lung edge map with an interactively drawn vertical line to enclose the lung region

The airway connection is inconsistently edge linked using the above steps. As a substitute, an interactively drawn line is added to all edge maps to consistently enclose the lung region, as seen in Figure 6.13. The interactively drawn vertical line near the airway connection encloses the lung region defined by the lung outline. Notably, when the segmented depth map is used to measure displacement, if the airway connection is segmented it should not affect the tidal volume or dynamic compliance, since it does not distend during ventilation.

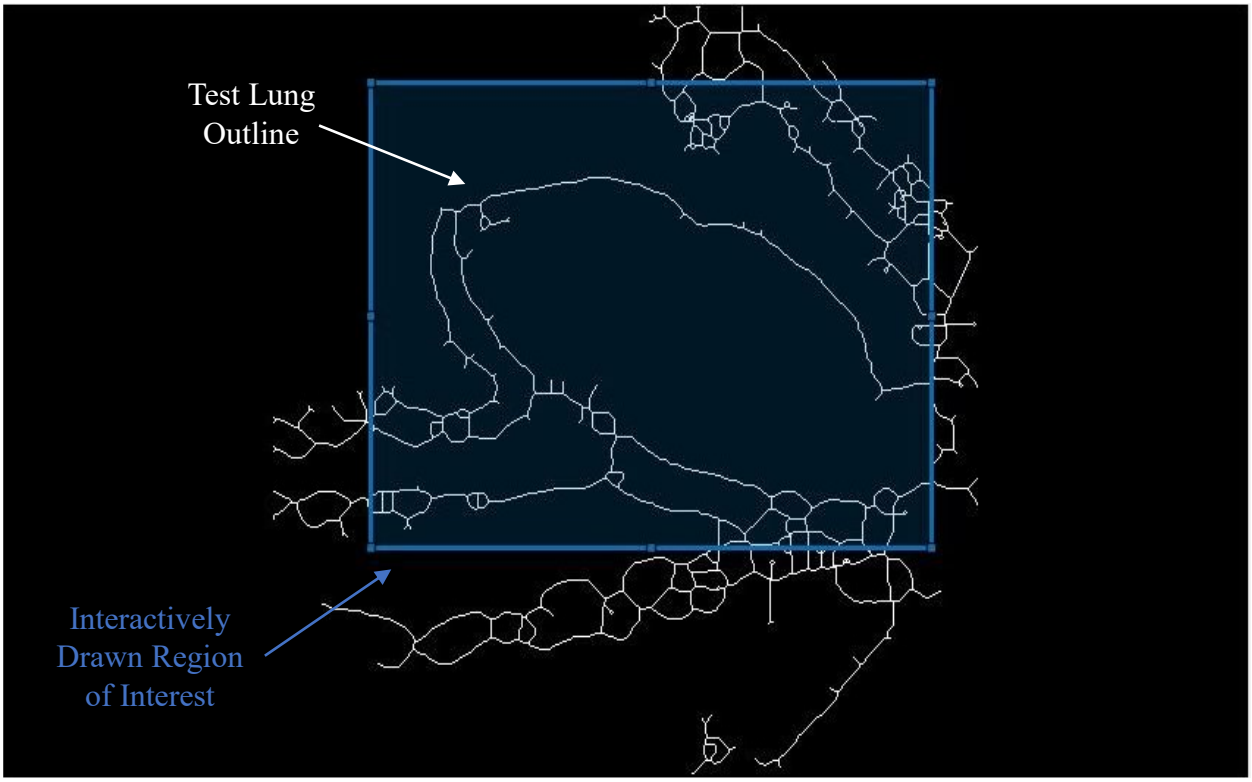


Figure 6.14 Annotated image of the ventilator test lung edge map with the interactively drawn cropping rectangle centered around the lung

Inside the ventilator test lung outline is the lung region that can be selected by taking the largest region in the complement of the edge map. The edge map was cropped interactively to simplify removing this search, removing insignificant regions, as seen in Figure 6.14. The interactively drawn cropping rectangle is centered on the lung region and is large enough to enclose the lung when fully distended. Notably, this region of interest is only drawn once, and is used for all edge maps.

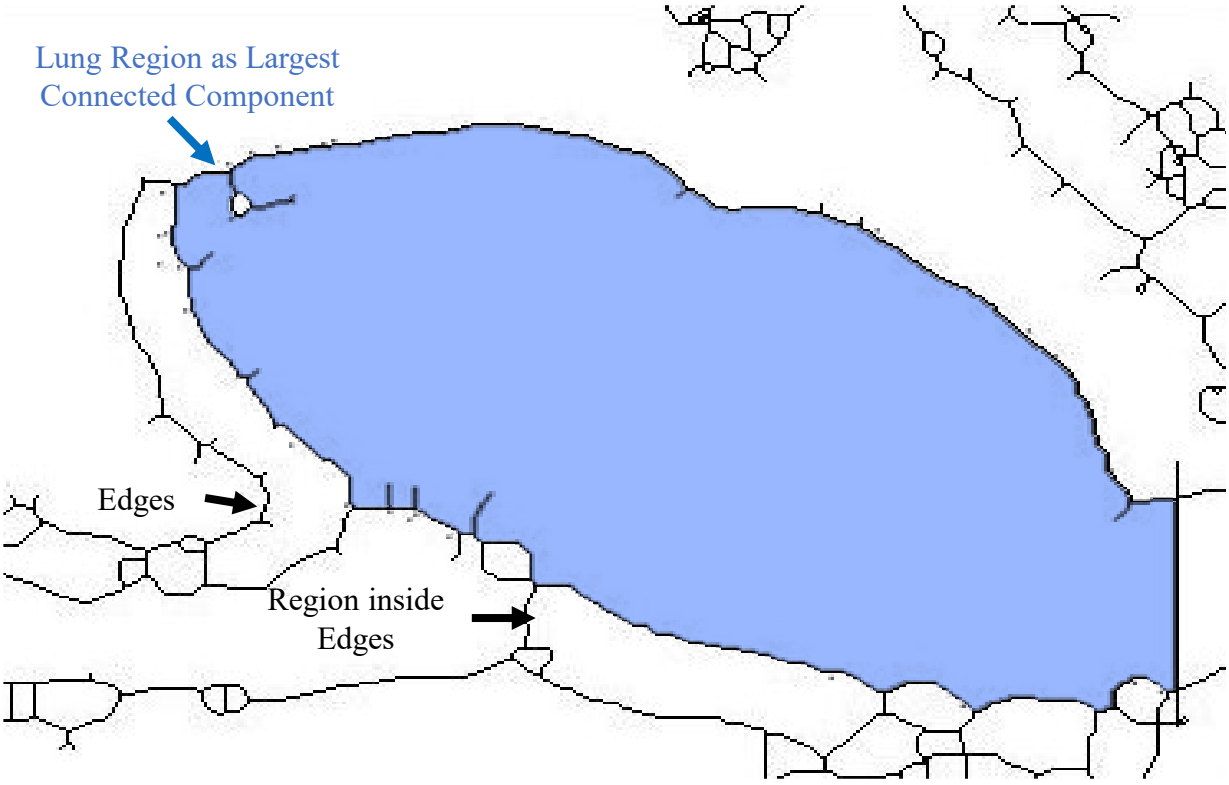


Figure 6.15 Annotated image of the complement of the edge map within the interactively cropped region

The lung was segmented as the largest region found within the complement of the edge map, as seen in Figure 6.15. The complement of the edge map is a binary map, where regions are white and black indicates borders. The labeled green region is the region with the most pixels assumed to be lung region. This region has various problems, such as spurs, and holes created during the edge detection process. These artifacts were filled using morphological operations.

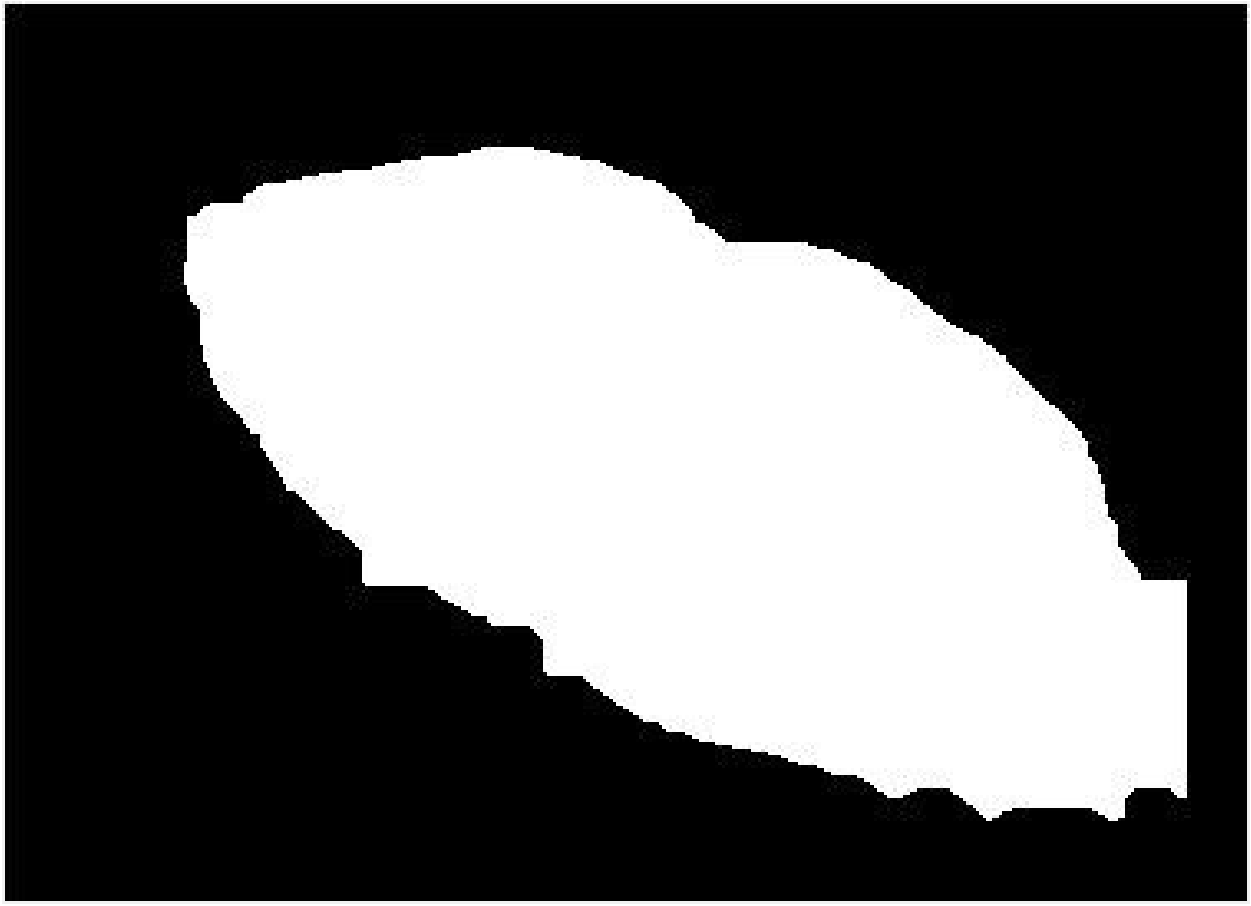


Figure 6.16 Image of the ventilator test lung binary map after hole filling post-processing

The spurs and holes in the binary map were filled using morphological closing, as seen in Figure 6.16. This binary map still had issues, namely the extrusion coming out of its perimeter that are not physically related to the lung. Outlier detection and temporally smoothing were performed to remove invalid frames and to remove these extrusions.

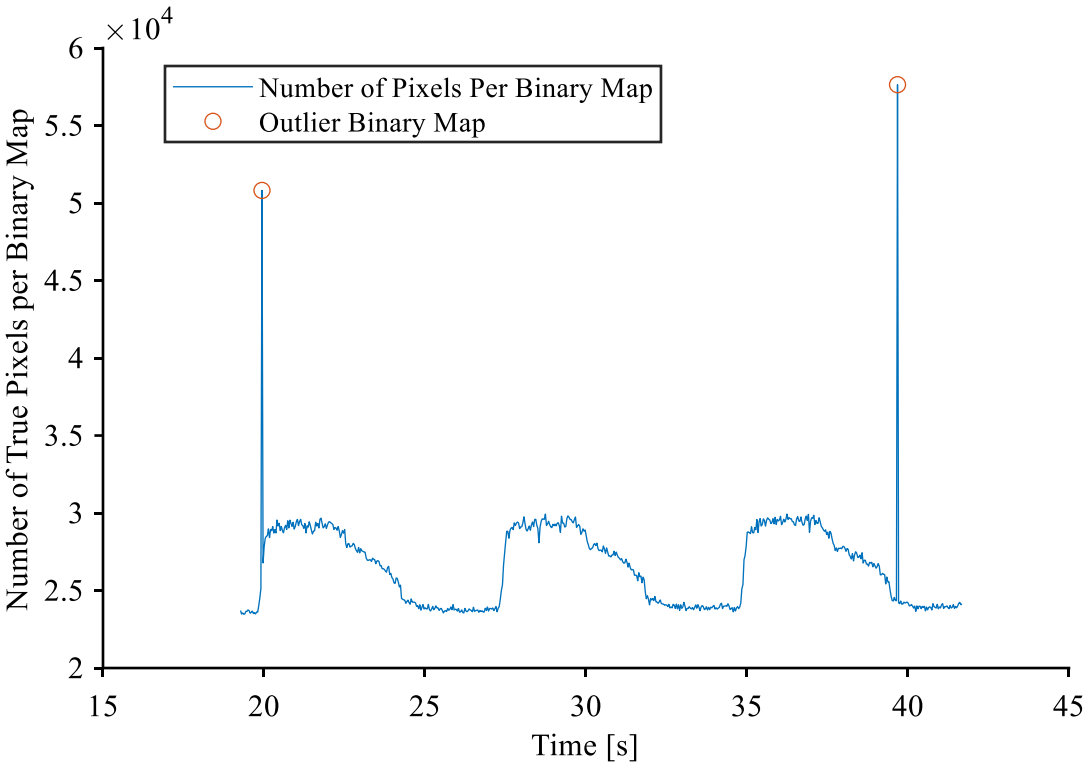


Figure 6.17 Plot of the area of each ventilator test lung region map for outlier detection

A Hampel digital filter was used to detect outlier frames based on the number of binary true pixels in each binary map. The number of true pixels per binary map formed a signal, as seen in Figure 6.17. The Hampel filter uses a moving window to calculate the local median and mean. A point is marked as an outlier if that data point is outside three times the local median from the mean. These outlier points were rejected before temporal smoothing of the binary maps.

After the outlier frames are discarded, the binary region maps are temporally averaged. The moving mean of each pixel is found, creating a set of grayscale images with the same number of frames and dimensions as the region maps. The grayscale moving mean images are binarized by a threshold value, between zero and one.

The result of this process are temporally smoothed binary maps of the lung region. This process can be interpreted as a low pass filter for each pixel between frames individually. The values of the moving mean images can be interpreted as the number of times a particular pixel is occupied within the span of the moving mean window. In this situation, the threshold is the cut-off for

how often a pixel must be occupied to remain in the binary map. If the sampling rate is known, this interpretation can be quantified in seconds.

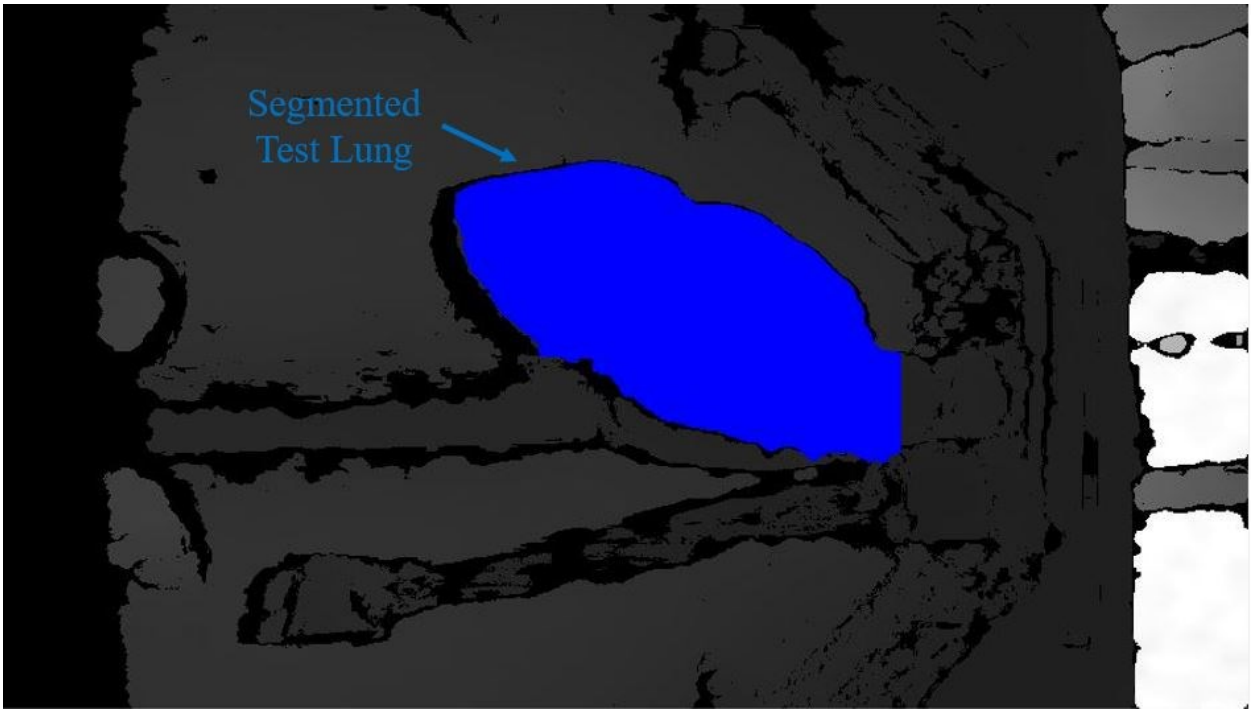


Figure 6.18 Annotated mage of the depth map of the ventilator test lung and EVLP chamber with the lung segmented

The temporally filtered binary maps are still cropped to the size of the interactively drawn rectangle. The last step is to insert the cropped binary maps into an image of the same size as the depth maps. These final binary maps were used to segment the ventilator test lung in the depth maps, as seen in Figure 6.18.

6.3.2 **Point Cloud Processing**

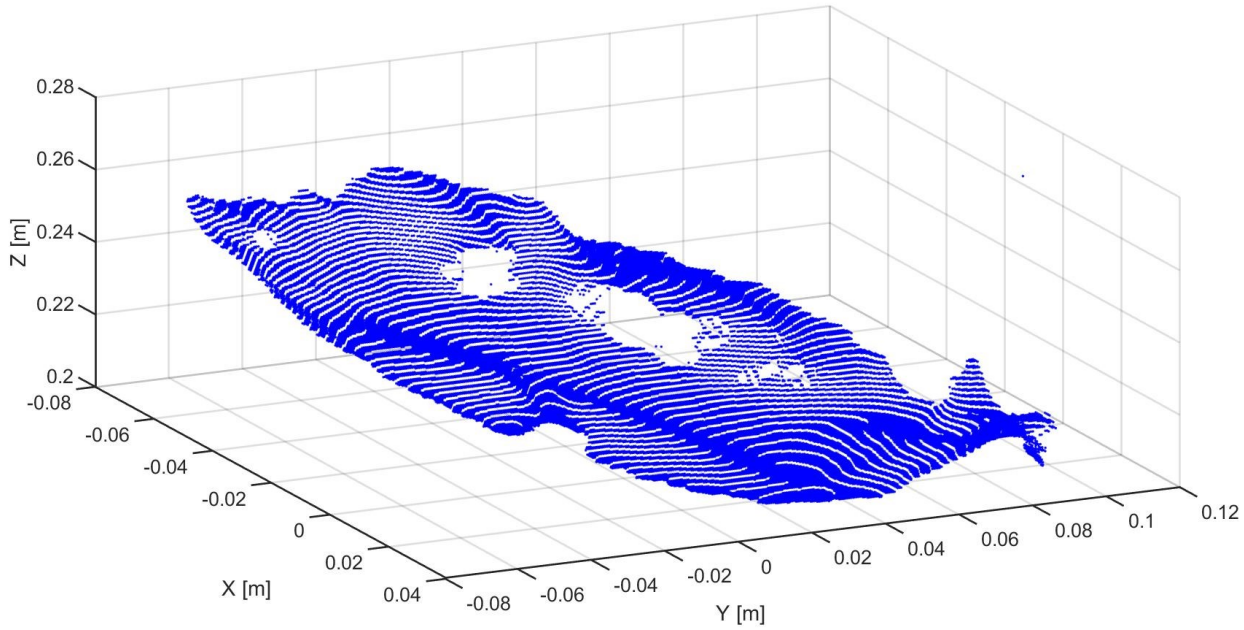


Figure 6.19 3D plot of the ventilator test lung point cloud

As described in Chapter 5, deprojection was used to convert the segmented depth map into a set of 3D points, called a point cloud. These points are along the surface of the test lung as seen in Figure 6.19. Each point corresponds to a single pixel in the depth map. The shape of the test lung can be extracted from this point cloud, to measure change in shape with respect to time. However, the point clouds are difficult to compare between image frames because they are composed of scattered points that are at different positions between frames. This scattered behavior can cause holes to form in the point cloud, along with unfavorable positioning of the Intel RealSense D435 causing occlusion, as seen in Figure 6.19. Furthermore, the coordinate system of the point cloud is not ideal for the measurement processing scheme.

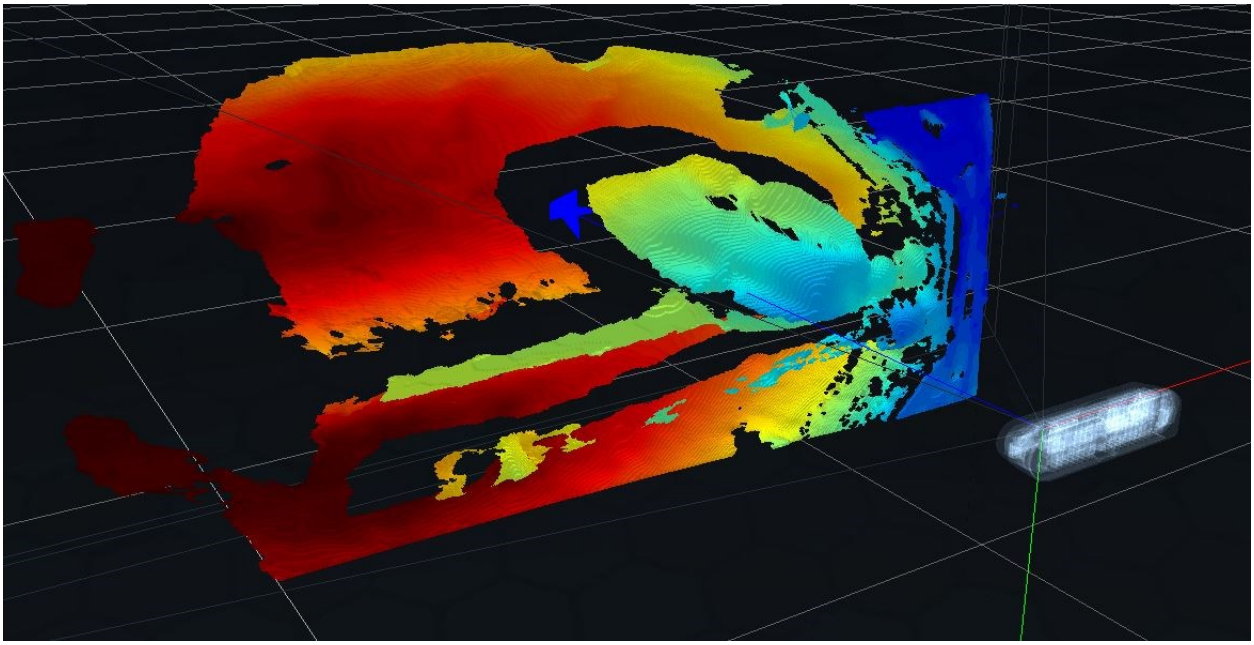


Figure 6.20 3D plot of the ventilator test lung and EVLP chamber point cloud in the Intel RealSense Viewer

The point cloud's coordinates are relative to the Intel RealSense D435, causing the top surface of the lung to have a lower height, or z coordinate, than its bottom, as seen in Figure 6.20. For the Divergence Theorem measurement method, ideally the point cloud's coordinate system is aligned with the height of the lung, measuring displacement in the positive z direction. If the z-axis datum of the coordinate system was the EVLP chamber floor the estimated volume would be closer to the actual lung volume. The z-axis datum can be arbitrary because it does not affect measurements of change such as tidal volume.

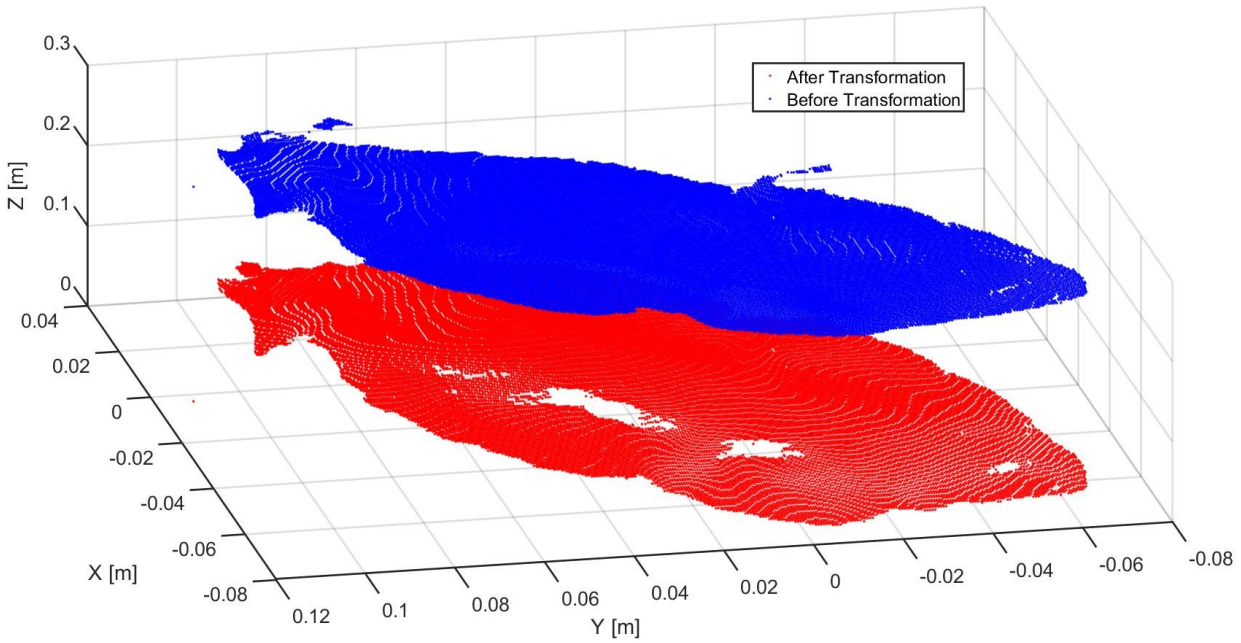


Figure 6.21 3D plot of the ventilator test lung point cloud before and after transformation

Three rigid transformations were performed on the ventilator test lung point clouds to move them into a new coordinate system, as seen in Figure 6.21. The point clouds were rotated 180º about *x*-axis, mirrored about the *YZ* plane, then translated the distance measured between the Intel RealSense D435 depth origin and the EVLP chamber floor. It was assumed that the Intel RealSense D435 and EVLP chamber floor were parallel since the tripod was leveled. Notably, the point clouds in Figure 6.21 are overly dense, as each have around 38,000 points.

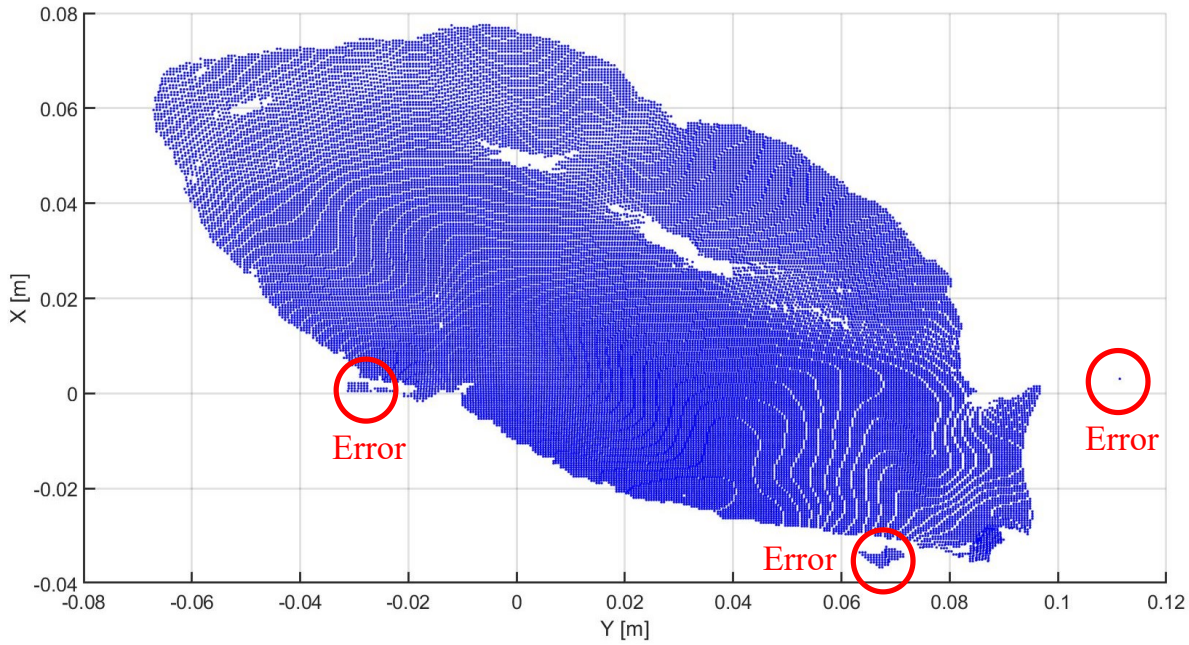


Figure 6.22 Annotated 3D plot of the ventilator test lung point cloud, with outliers annotated

The point clouds are dense and have some stray points circled in red, as seen in Figure 6.22. This high point resolution is expected to improve surface reconstruction only marginally, while drastically increasing computational time and memory cost for any future steps. Both problems were solved by downsampling to reduce the number of points and remove outliers like the stray points.

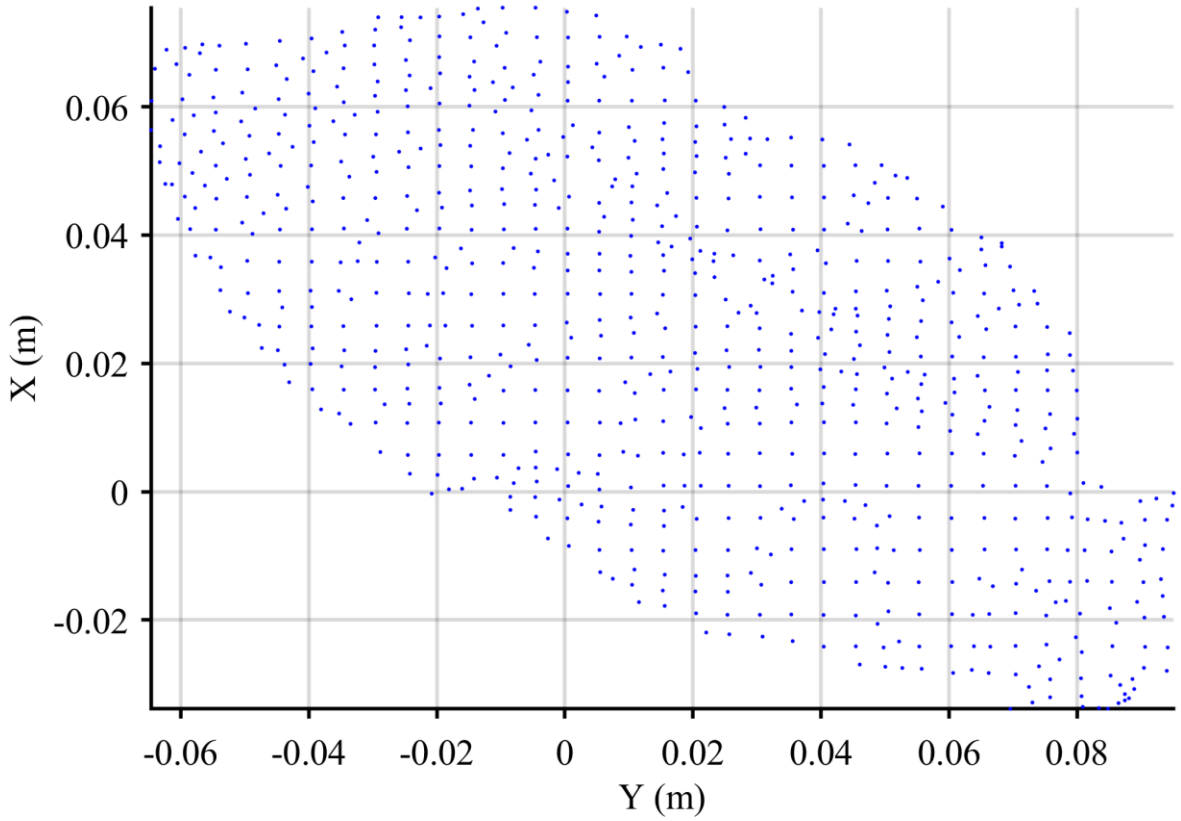


Figure 6.23 3D plot of the ventilator test lung point cloud after 3D box averaging filtering

A 3D box averaging filter was used to downsample the ventilator test lung point clouds, as seen in Figure 6.23. The box averaging filter splits the *XYZ* domain into cubes, called voxels, then returns the average coordinates of all the points within each voxel. It has the properties of a low pass filter in the spatial domain, smoothing the surface. The desired effect was to reduce the number of points in each point cloud, about 38,000 to 800 points, while preserving the shape of the ventilator test lung surface. Also, small groups of points, called clusters were removed from the point cloud to remove the stray points if they remained after box average filtering. These steps prepared the point clouds to extract a surface model of the ventilator test lung for displacement measurement.

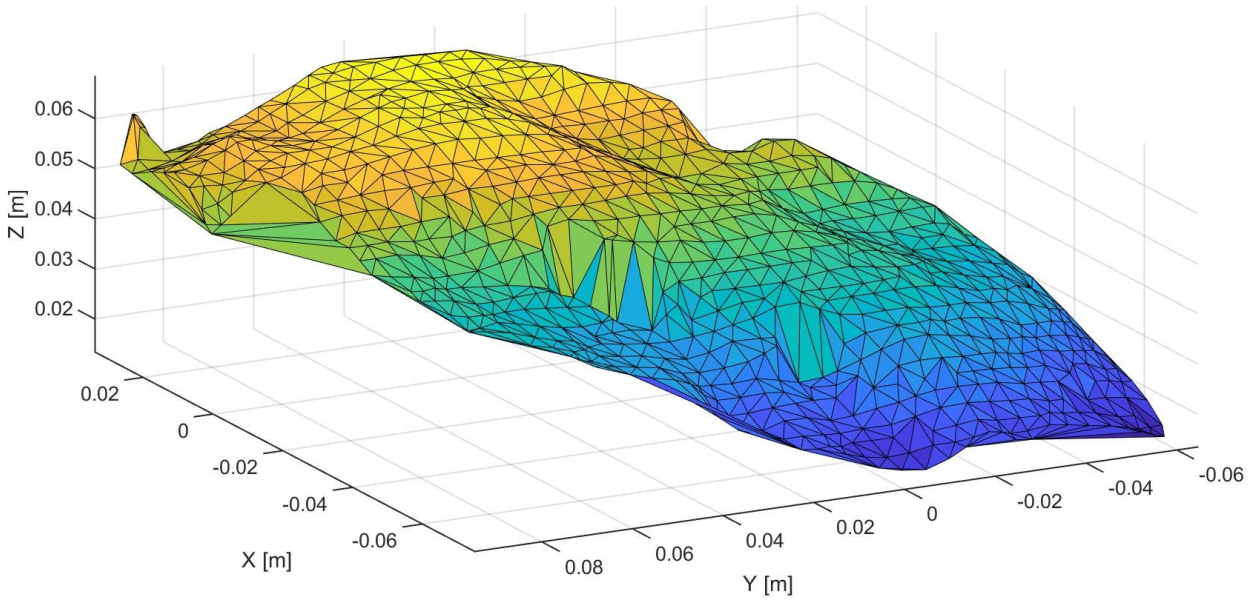


Figure 6.24 3D plot of the ventilator test lung Delaunay triangulation surface mesh

The point clouds were meshed using Delaunay triangulation to obtain a model of the surface of the ventilator test lung, as seen in Figure 6.23. Specifically, the mesh was found using the x and y coordinates of the 3D points of the point cloud.

The point clouds were not interpolated to obtain surface maps, as described in Chapter 5, because the ventilator test lung moved during ventilation. As a result, the regional measurements were not useful since they do not track the lung's surface. Therefore, the interpolation step was skipped because it was unnecessary to calculate the scalar plethysmography measurements.

6.3.3 **Plethysmography Measurements**

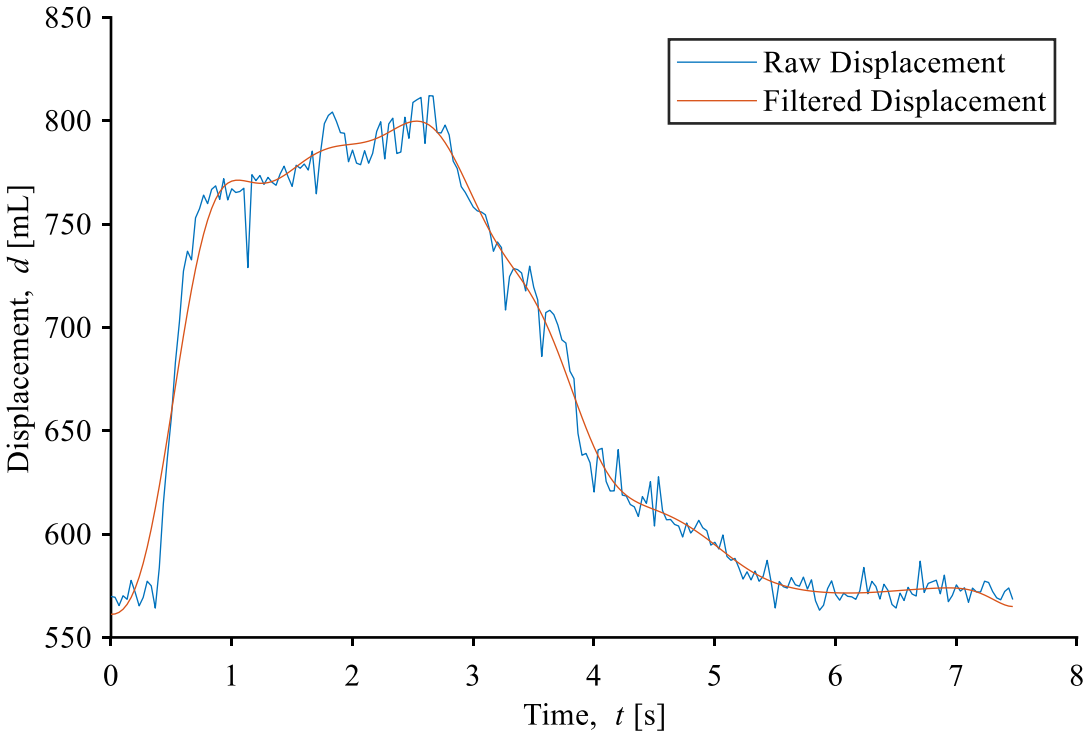


Figure 6.25 Plot of the ventilator test lung displacement before and after signal filtering

The volume between each surface mesh and the XY plane was measured using the integration method derived from the Divergence Theorem described in Chapter 5 [32]. This produced a raw volume signal for each experiment configuration, as seen in Figure 6.25 for one cycle. The volume signal was denoised and smoothed using a low pass frequency filter and a Savitzky-Golay digital filter to obtain a filtered signal, as seen in Figure 6.25. Notably, the low pass filter cut-off frequency was set just above the ventilation rate of 8 breaths per minute.

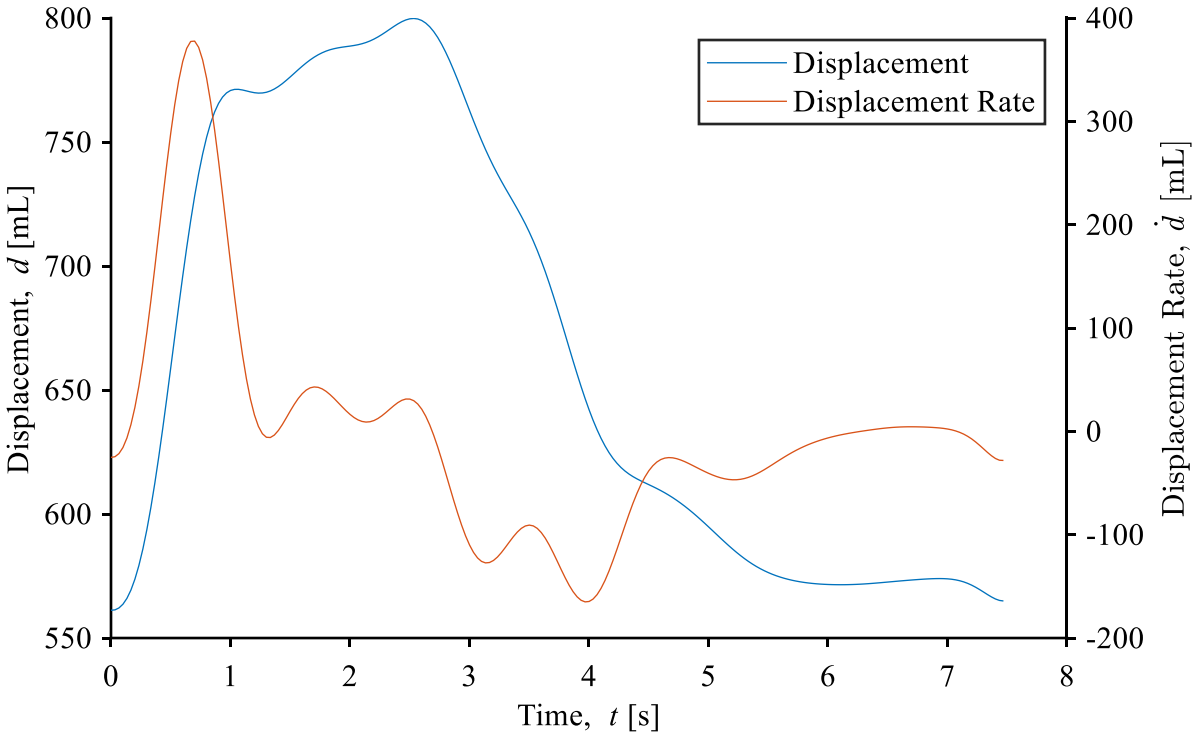


Figure 6.26 Plot of the ventilator test lung displacement and displacement rate

The displacement signal was used to derive other measurements, such as an analogous signal to flow rate, as described by [35]. The first derivative of the displacement curve was found using a FIR differentiator filter, as seen in Figure 6.26. The differentiator filter was designed by defining the filter order, pass band frequencies, stop band frequency, and sampling rate as inputs for the *designfilt*() MATLAB function. The differentiator filter gives better results than finding the instantaneous difference between digital measurements, which amplifies noise [127]. Also, the displacement signal was used to calculate the tidal parameters described in Chapter 5 based on their definitions.

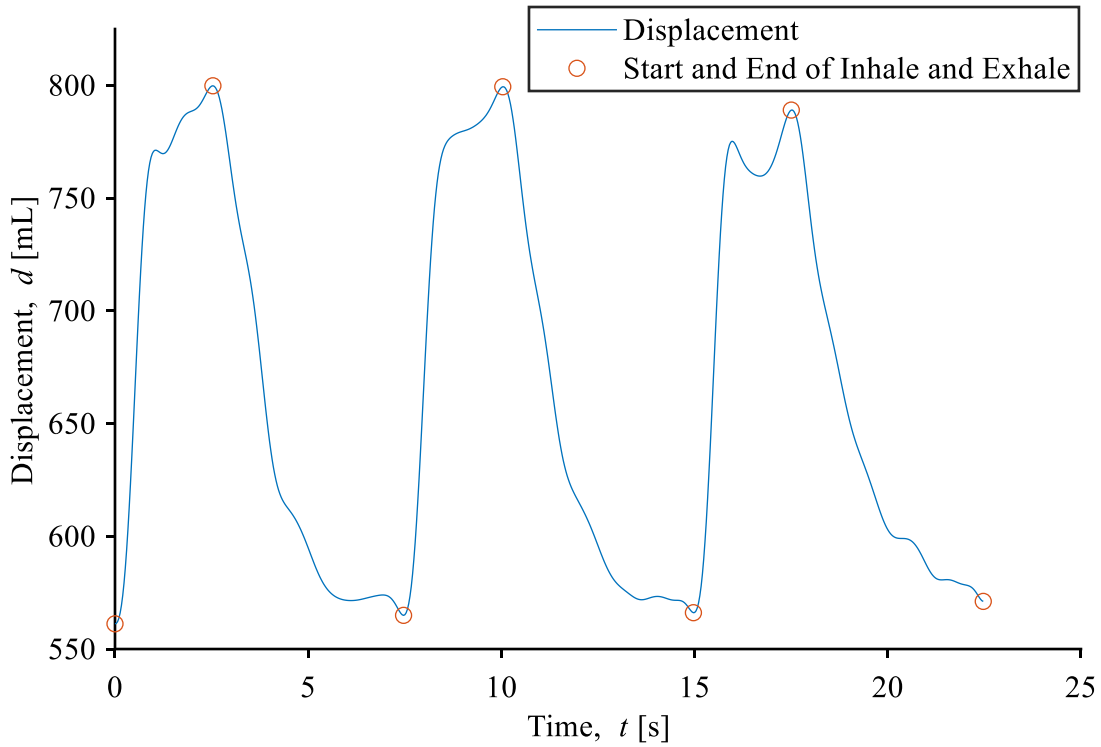


Figure 6.27 Plot of the ventilator test lung ASV displacement segmented for the inhale and exhale points from case 1 for the first three breaths

Mechanical ventilation imitates the inhale and exhale phases of tidal breathing. The points where inhale transitions to exhale, and vice versa, can be found as the local extrema points of the displacement signal. These points were found using the MATLAB function *findpeaks*() which uses the zero-crossing method [128], along with several outlier detection methods. These inhale-exhale points can be used to segment the displacement signal into breaths, allowing the measurement of breath specific metrics including tidal volume, and dynamic compliance.

Figure 6.27 shows the ventilator test lung displacement curve for the first three breaths with the identified inhale-exhale points. The first breath can be segmented as the displacement curve between the first to third inhale-exhale points. Similarly, the first inhale phase is the displacement between the first and second inhale-exhale points, and the first exhale phase is between the second and third points. These segmentation processes can be replicated with the knowledge that every other point is the start of inhale, or end of exhale, while every even order point is the end of inhale or start of exhale.

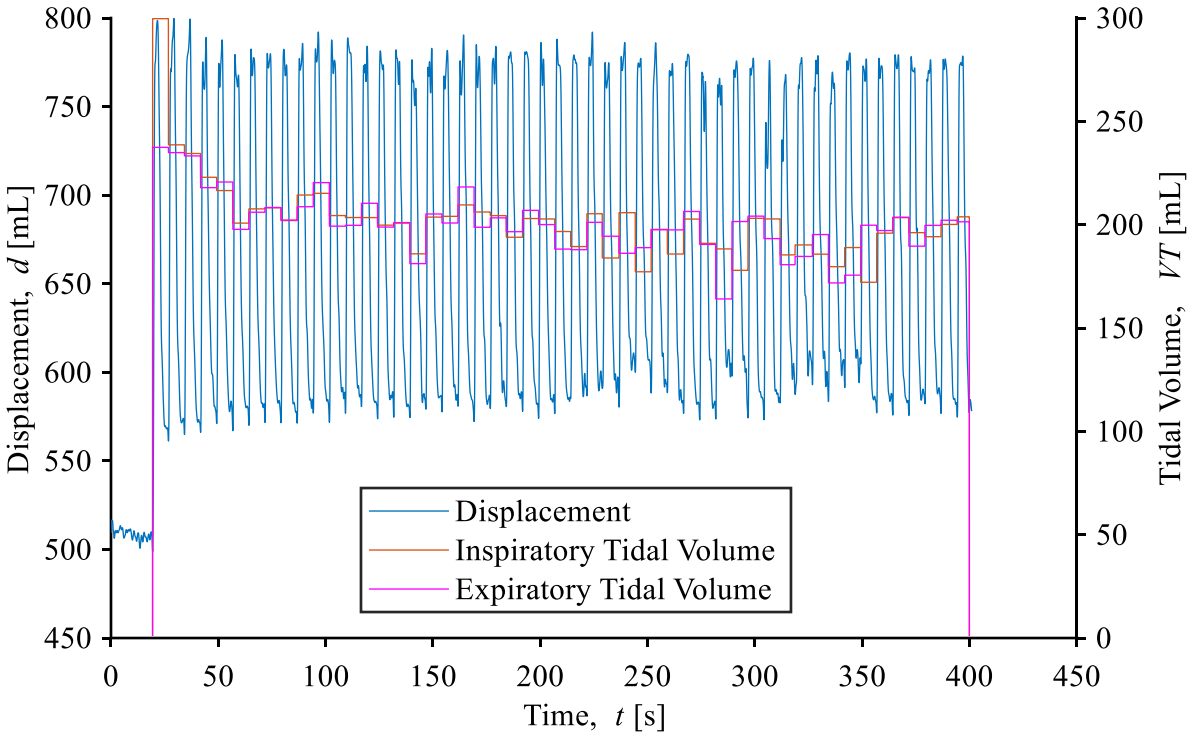


Figure 6.28 Plots of the ventilator test lung ASV displacement and tidal volume from case 1

The inspiratory and expiratory tidal volume can be found as the displacement difference of pairs of inhale-exhale points, as seen in Figure 6.28. Notably, the first tidal volume is an outlier because the sudden increase in pressure at the start of ventilation. Afterwards, the ventilator test lung is ventilated at a higher-pressure range. Tidal volume can be used to find dynamic compliance with the EVLP pressure measurements if they are synchronized with the Intel RealSense D435.

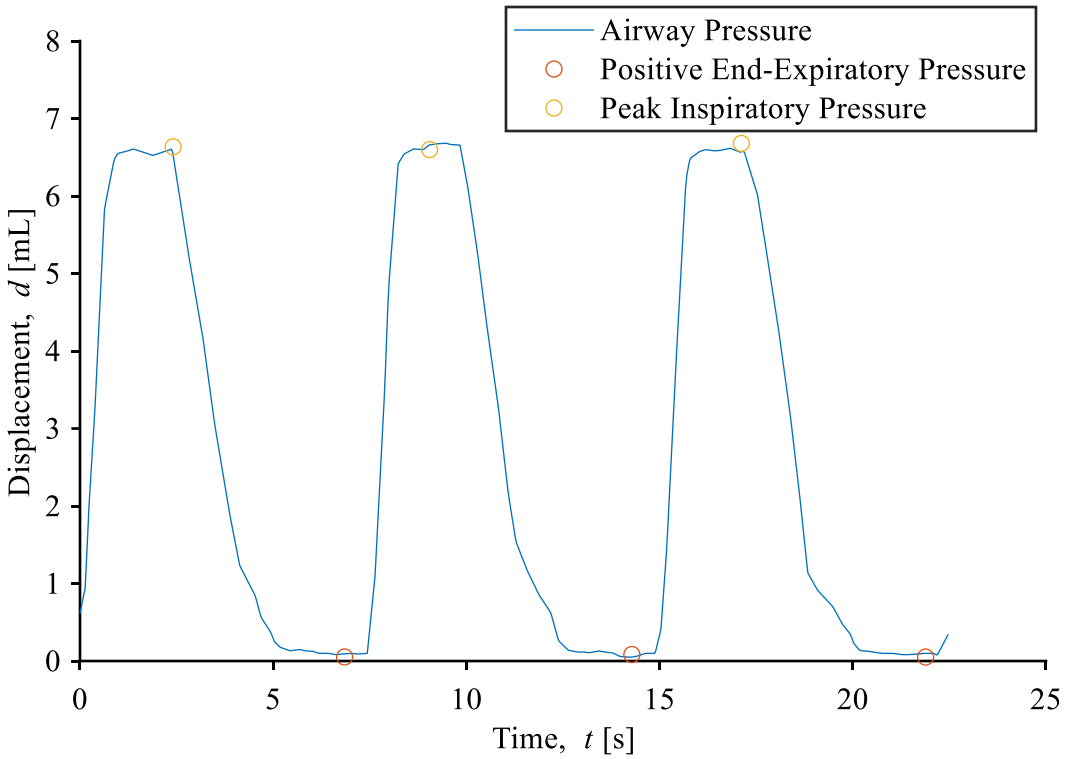


Figure 6.29 Plot of the EVLP airway pressure, PIP, and PEEP

The EVLP and Intel RealSense D435 measurements were synchronized by interpolating the EVLP to the Intel RealSense D435 sample times. Both sets of measurements were recorded with timestamps, with the same datum. The EVLP was interpolated instead of the Intel RealSense D435 measurements because the Intel RealSense D435 had a higher sampling rate, and the Intel RealSense D435 recorded measurements before and after the EVLP for each experiment case.

Synchronizing the measurements allows them to be compared. Also, the interpolated EVLP pressure measurements were used to calculate dynamic compliance from the Intel RealSense D435 tidal volume. *PIP* and *PEEP* were found based on the Intel RealSense D435 displacement signal inhale-exhale points. *PIP* and *PEEP* were found as the EVLP airway pressure at the inhale-exhale points from the Intel RealSense D435 displacement signal, as seen in Figure 6.29. Dynamic compliance was measured using (3-7). The tidal parameters listed in Chapter 5 were calculated for each experiment case from the displacement and displacement rate signals including inspiratory time, *PTEF*, and IE ratio.

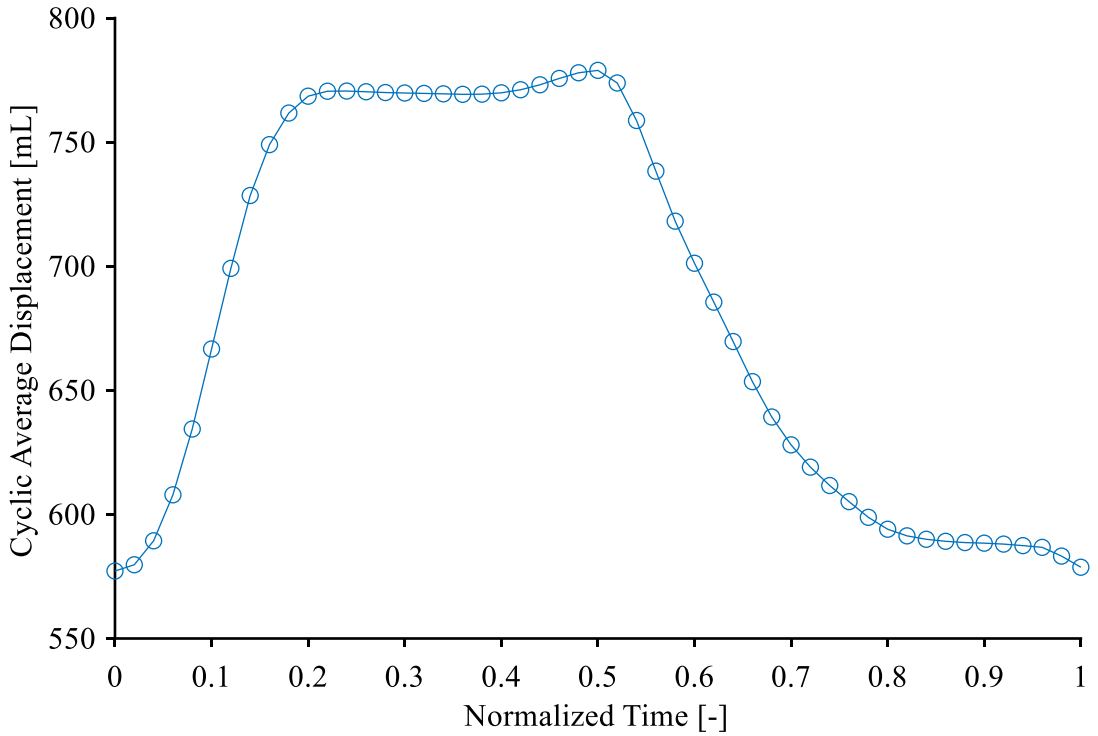


Figure 6.30 Plot of the cyclic average displacement of the porcine lung

As previously mentioned, the baseline performance of the porcine lung could be measured as a cyclic average of the displacement curve, as seen in Figure 6.30. This processing scheme averages all the segmented breaths, into an averaged volume relative to a normalized time within a cycle. The displacement average cycle was found using the *create_avg_cycle()* function in the appendix.

The cyclic averaging was performed by interpolating displacement of each cycle, at the same locations within each cycle. The time between points is normalized by the period of each cycle. This means, the first sample location for each cycle is at the same normalized time relative to the start of each cycle. The displacement of each normalized point is averaged across all cycles, into one cyclic average cycle.

6.3.4 **Surface Measurements**

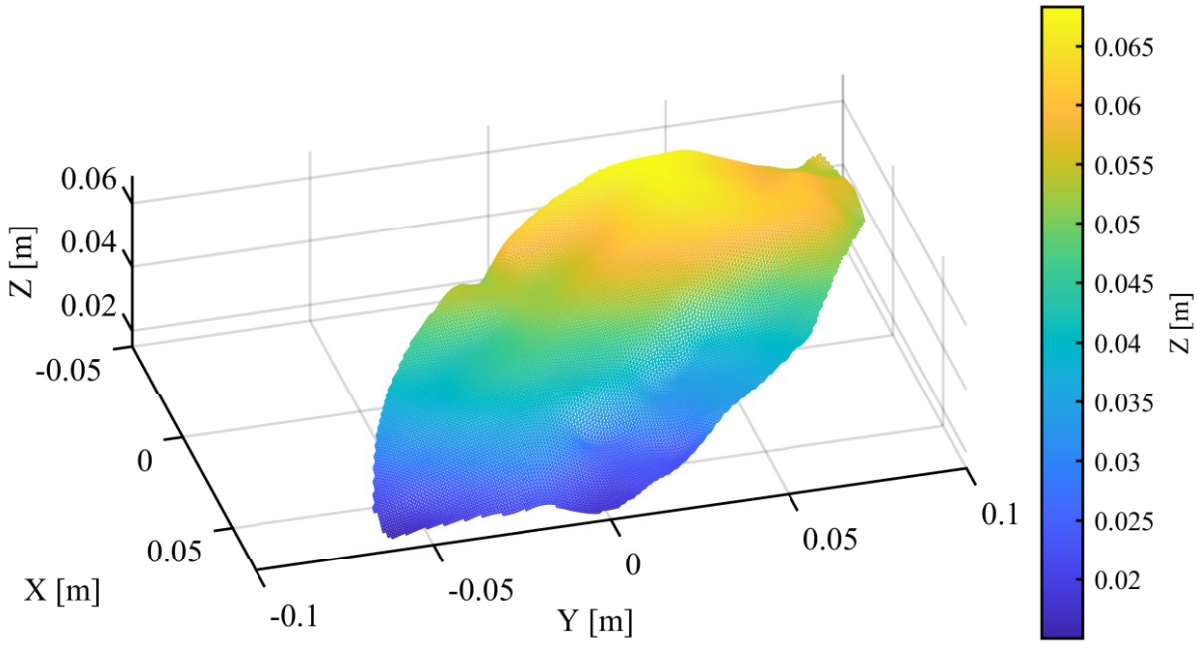


Figure 6.31. 3D plot of surface map of the ventilator test lung

The surface maps of the ventilator test lung were found using the method described in Chapter 5.4 , as seen in Figure 6.31. They were used to find the surface cyclic averages in similar method to the cyclic average displacement in Figure 6.30. Also, the surface maps were used to calculate surface tidal displacement and surface dynamic compliance. However, these regional measurements were invalid.

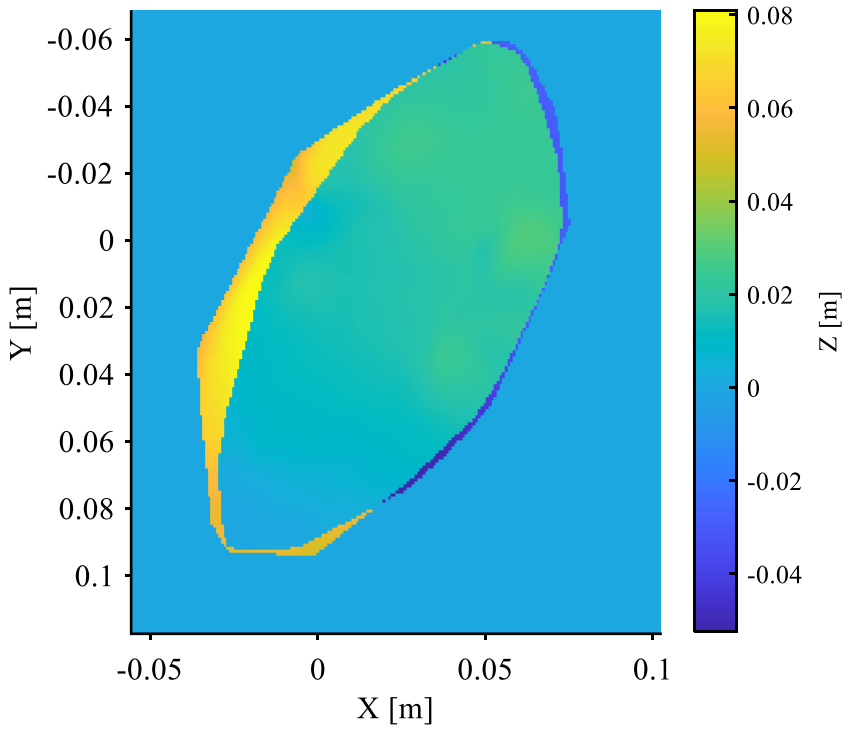


Figure 6.32. Plot of the surface tidal displacement of the ventilator test lung

As previously mentioned, the ventilator test lung moved during ventilation. Since active stereo vision does not track the lung's surface, the regional measurements from the surface maps were invalid. Specifically, the surface tidal displacement and dynamic compliance had multiple regions, outlining where the lung was at the start and end of inhale, as seen in Figure 6.32. The ventilator test lung moves to the yellow region at peak inhale, and then moves into the blue region at the end of exhale. The regions between them are common between the ends of inhale and exhale. As a result, the common region is blurred by the ventilator test lung moving across it, distorting the surface shape. Also, the inhale and exhale exclusive regions are global and local peaks and troughs in the regional measurements. Therefore, these regional measurements and peak detection are not informative.

6.4 Comparison of Measurements

The ASV method was evaluated by comparing its plethysmography measurements with the EVLP system. This analysis evaluated measurement distribution, correlation, and agreement. The difference in measurement distribution mean and standard deviation provides a preliminary error analysis of the ASV method. Correlation and linearity were evaluated to identify if the two methods measured the same changes in the ventilator test lung at the same time. High correlation would suggest the ASV method is measuring the ventilator test lung. Agreement was measured to determine if the ASV measured these changes with the same accuracy as the EVLP. Also, high agreement may suggest the methods are interchangeable for plethysmography.

6.4.1 Reference and Competing Measurements

The plethysmography measurements were used to compare the ASV and EVLP measurement methods. The differences in the tidal volume and dynamic compliance measurements were the focus of the studies. Respiratory cycle timing parameters such as inspiratory time were compared too. However, displacement rate was not compared with the airway flow rate because they are expected to have significantly different magnitudes. Lastly, pressure measurements were not compared because the ASV method only measures surface displacement. Notably, parameters such as *IE50* and *tTEF50* were excluded because they could not be determined from the EVLP system measurements.

The EVLP method had erroneous dynamic compliance measurements that were several magnitudes greater than the recorded tidal volume and pressure measurements. These measurements were ignored and recalculated in MATLAB using the recorded tidal volume, pressure, and the inhale-exhale points found from the breath state signal.

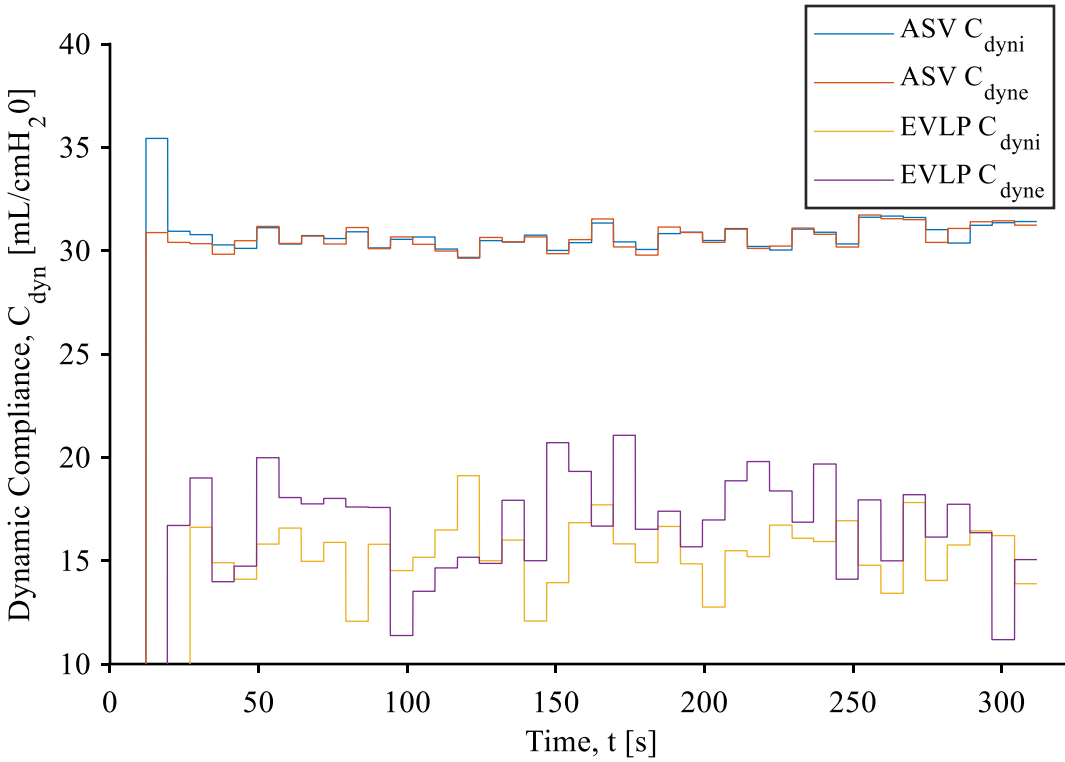


Figure 6.33 Plot of the ASV and EVLP dynamic compliance of the ventilator test lung from case 6

The ASV and EVLP measurements were taken with different sampling rates. To compare them, the ELVP measurements were temporally aligned and interpolated at the same time stamps as the ASV measurements. This process provided paired measurements for all plethysmography metrics, one pair for each respiratory cycle. Any pairs with not-a-number (Nan) or infinity based on MATLAB arithmetic were removed from the comparison analysis. Lastly, the EVLP measurements were used as the reference values in all comparisons.

All experimental cases were processed using this method. In some cases, the ASV method was found to have lower dispersion than the EVLP measurements, as seen in Figure 6.33. Cases 4, 5, 6, and 8 showed this behavior. Also, the ASV measurements were found to be immune to erroneous pressure measurements.

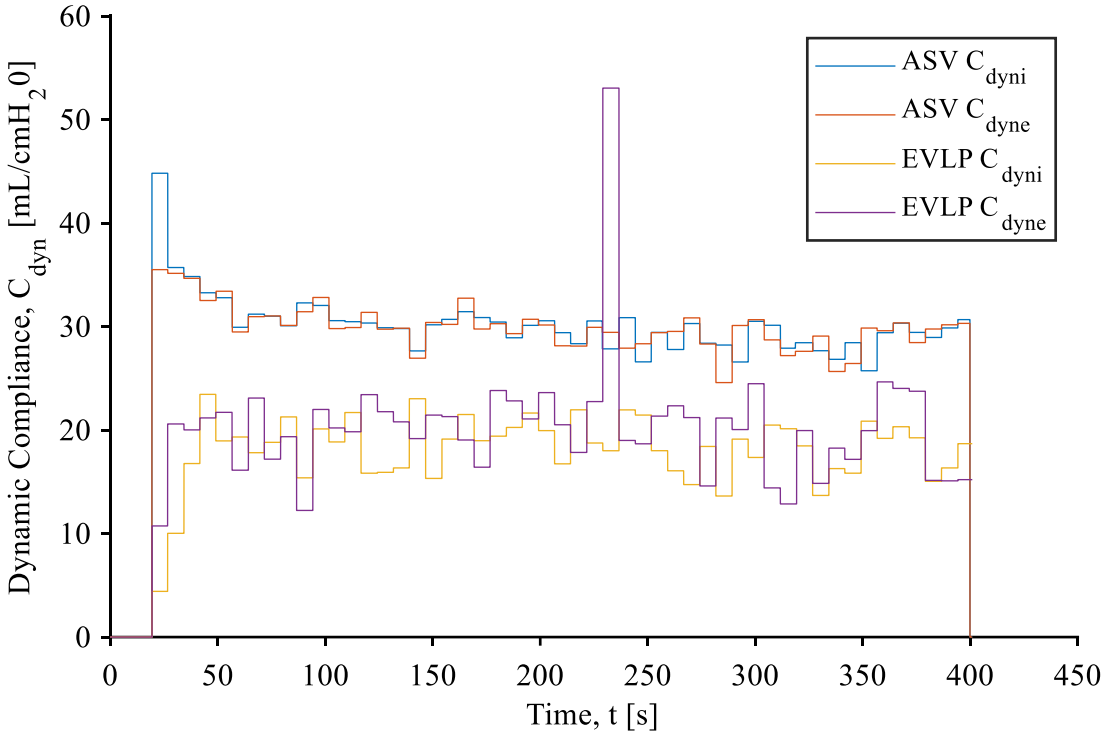


Figure 6.34 Plot of the ASV and EVLP dynamic compliance of the ventilator test lung from case 1

Outliers from inaccurate flow rate and pressure measurements are ignored by the ASV method since it measures physical displacement. The outlier peak in the inspiratory dynamic compliance of the EVLP was ignored by the ASV method, as seen in Figure 6.34. This outlier was caused by an instantaneous spike in pressure.

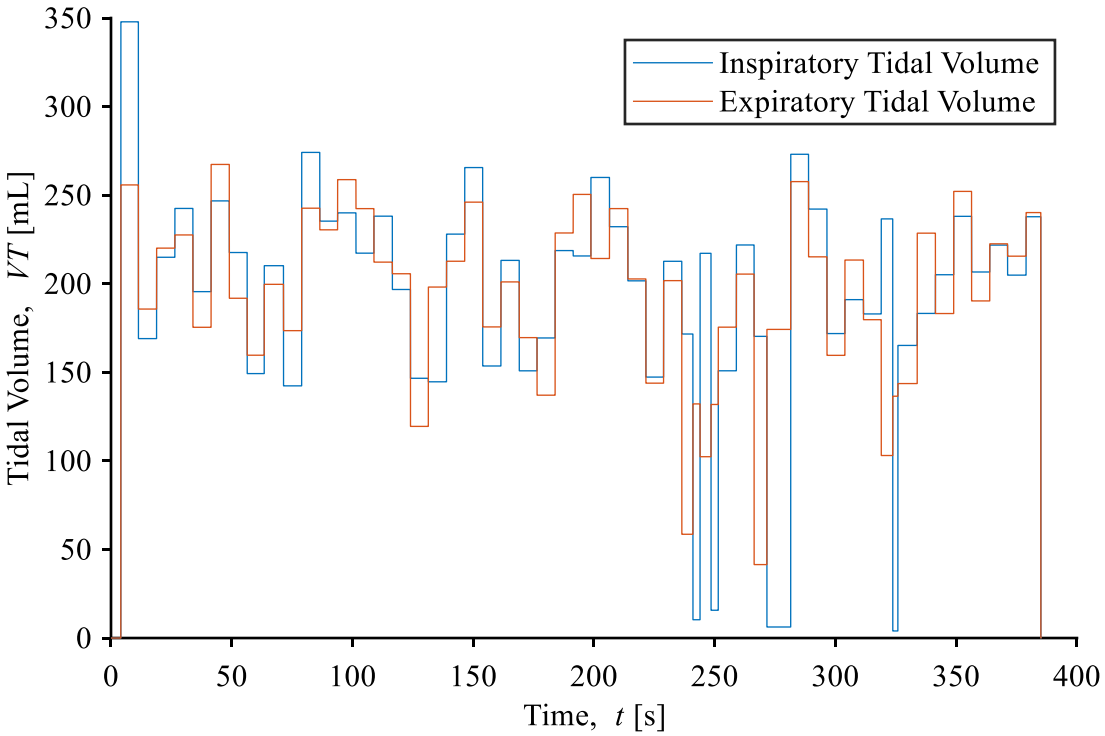


Figure 6.35 Plots of the ventilator test lung tidal volume from case 3

Case 3 had an erroneous displacement signal that invalided all other metrics, as seen in Figure 6.35. Specifically, the erratic behavior of the displacement signal led to poor respiratory cycle segmentation that impacted all measurements. It is hypothesized that this was caused by erroneous depth map segmentation. Measurements from case 3 were excluded from the comparison analysis.

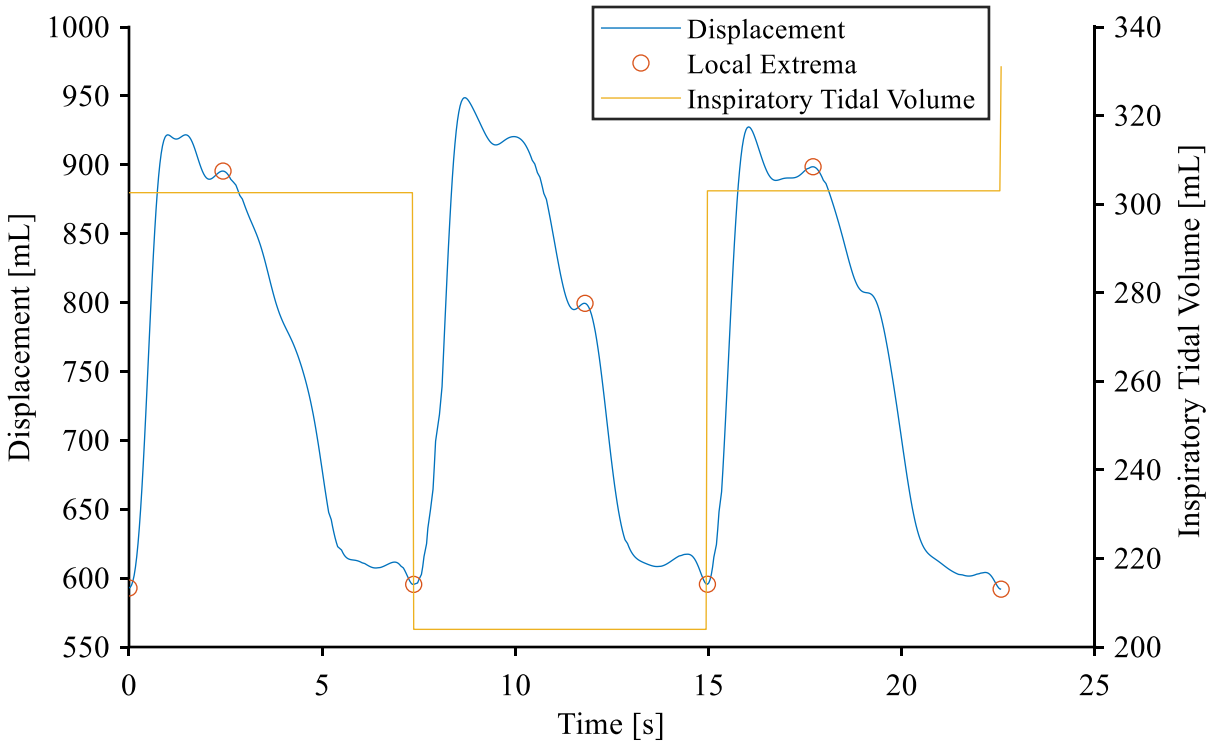


Figure 6.36 Plots of the ventilator test lung tidal volume from case 9

Case 9 was another case with noticeable ASV measurement error. The respiratory cycle segmentation was erroneously caused bimodal tidal volume measurements, as seen in Figure 6.36. Specifically, a lower secondary peak was detected for some breaths in the displacement signal. Also, the EVLP measurements had issues too.

6.4.2 **Measurement Distribution**

The ASV and EVLP measurement distributions centers and variance were compared as a preliminary error analysis. Mean was assumed to be an appropriate center metric for comparison. Also, the methods' measurement distribution were compared relative to normality using kurtosis and skewness scores and checking for normality using the Shapiro-Wilk hypothesis test. Lastly, the ASV and EVLP measurement distributions were visualized to observe their behaviour and validate any observations from the distribution properties.

Table 6.2 Mean and standard deviation of the ventilator test lung ASV and EVLP measurements of case 1

Parameter	Mean			Standard Deviation		
	ASV	**EVLP**	**Error [%]**	**ASV**	**EVLP**	**Error [%]**
Inspiratory Tidal Volume, *VTi* [mL]	226.05	136.52	65.58	35.91	29.56	21.48
Expiratory Tidal Volume, *VTe* [mL]	225.18	145.82	54.42	33.95	37.99	-10.63
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	29.52	17.62	67.54	3.16	3.59	-11.98
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	29.28	19.22	52.34	2.68	4.79	-44.05
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	0.75	0.58	29.31	0.16	0.08	100.00
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	1.48	1.35	9.63	0.67	0.39	71.79
Inspiratory Time, *tI* [s]	2.61	2.48	5.24	0.35	0.15	133.33
Expiratory Time, *tE* [s]	4.88	5.01	-2.59	0.43	0.17	152.94
Total Time, *tTot* [s]	7.48	7.49	-0.13	0.32	0.13	146.15
Inspiratory Total Time Ratio, *tITot*	0.35	0.33	6.06	0.05	0.02	150.00
Inspiratory Expiratory Time Ratio, *tIE*	0.54	0.50	8.00	0.12	0.04	200.00
Respiratory Rate, *RR* [bpm]	8.03	8.02	0.12	0.38	0.14	171.43

The mean and standard deviation of the ASV and EVLP measurements were measured to evaluate the ASV method's performance relative to the EVLP, Table 6.2. The ASV tidal volume and dynamic compliance means are significantly larger than the EVLP measurements. On the other hand, the timing metrics have low error. The mean error of the pressure metrics is unexpected because they are derived from the same signal, measured by the EVLP. Also, the low error of the timing parameters could indicate that the ventilator test lung is distending in sync with ventilation, and the ASV method measures these changes within reasonable accuracy. Notably, the ASV method for most parameters has more variance than the EVLP, except for tidal volume and dynamic compliance.

Table 6.3 Kurtosis and skewness of the case 1 ASV and EVLP measurement of the ventilator test lung

Parameter	ASV Excess Kurtosis	EVLP Excess Kurtosis	ASV Skewness	EVLP Skewness
Inspiratory Tidal Volume, *VTi* [mL]	2.13	23.91	0.46	1.09
Expiratory Tidal Volume, *VTe* [mL]	-0.76	8.74	-0.60	3.38
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	7.46	7.12	0.57	-0.94
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	-0.26	2.35	0.54	-2.17
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	1.02	2.86	0.27	0.85
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	-1.21	-0.05	0.28	0.28
Inspiratory Time, *tI* [s]	2.97	4.87	-0.88	-1.65
Expiratory Time, *tE* [s]	-1.19	31.99	0.21	4.89
Total Time, *tTot* [s]	15.42	-0.91	3.91	0.50
Inspiratory Expiratory Time Ratio, *tIE*	1.21	2.71	0.35	0.53
Inspiratory Total Time Ratio, *tITot*	15.37	16.32	-1.23	-2.90
Respiratory Rate, *RR* [bpm]	6.29	37.41	1.33	4.96

Kurtosis and skewness of both methods' measurement distributions for all parameters and experiment cases were found to quantify they shape. Kurtosis is a measure of tailedness, or how the distribution at its extremes deviates from the normal distribution [129]–[131]. Distributions with higher kurtosis have thinner but wider tails with a higher probability of outliers [130]. A standard normal distribution has a kurtosis of three, for convenience, scores can be offset so a normal distribution has a score of zero. This score convention is called excess kurtosis [129].

Skewness is a measure of asymmetry of a distribution [131]. A standard normal distribution has perfect symmetry, with a skewness score of zero where the mean, mode, and median align with the center of the distribution. A right skew, with a positive skewness, has a right tail and a left peak. The opposite is true for a left skew, with a negative skewness. For the right and left skew distributions, the mode is found at the peak, the median at the center of the range, and the mean near the tail [131].

In general, the kurtosis and skewness of measurements from case 1 are representative of the other cases, as seen in Table 6.3. Based on the skewness scores, most of the parameters from the ASV method are approximately symmetric, with their mean at the center of their distribution. Some examples are the expiratory dynamic compliance distributions with a skewness of 0.54.

Also, their excess kurtosis scores suggest slight tailedness. For expiratory dynamic compliance, excess kurtosis was -0.26. However, the EVLP does not share the same distribution shape as the ASV for most parameters.

The ASV had kurtosis and skewness scores closer to normality than the EVLP. However, the EVLP seems to be more prone to outliers since it has a higher kurtosis. This is unexpected since the EVLP standard deviations are narrower than the ASV method. Also, the ASV and EVLP measurements have less skew than the EVLP for the same metrics. Notably, there are several parameters with extreme kurtosis and skewness scores such as inspiratory time. In general, these results are difficult to derive conclusions from. To simplify this evaluation, the distributions were categorized based on their kurtosis and skewness scores for tailedness and symmetry.

Table 6.4 Kurtosis and skewness of the case 1 ASV and EVLP measurement of the ventilator test lung

Parameter	**Tailedness**		**Symmetry**	
	ASV	**EVLP**	**ASV**	**EVLP**
Inspiratory Tidal Volume, *VTi* [mL]	Platykurtic	Platykurtic	Zero skew	Right-Tailed
Expiratory Tidal Volume, *VTe* [mL]	Mesokurtic	Platykurtic	Zero skew	Right-Tailed
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	Platykurtic	Platykurtic	Zero skew	Zero skew
Expiratory Dynamic Compliance, C_{dyne} [mL/cm$H_2$0]	Mesokurtic	Platykurtic	Zero skew	Right-Tailed
Inspiratory Time, *tI* [s]	Platykurtic	Platykurtic	Zero skew	Left-Tailed
Expiratory Time, *tE* [s]	Leptokurtic	Platykurtic	Zero skew	Right-Tailed
Total Time, *tTot* [s]	Platykurtic	Leptokurtic	Right-Tailed	Zero skew
Inspiratory Expiratory Time Ratio, *tIE*	Platykurtic	Platykurtic	Zero skew	Zero skew
Inspiratory Total Time Ratio, *tITot*	Platykurtic	Platykurtic	Left-Tailed	Left-Tailed
Respiratory Rate, *RR* [bpm]	Platykurtic	Platykurtic	Right-Tailed	Right-Tailed
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	Platykurtic	Platykurtic	Zero skew	Zero skew
Time to Peak Tidal Expiratory Flow, *tPTEF* [*s*]	Leptokurtic	Mesokurtic	Zero skew	Zero skew

Distributions can be categorized as mesokurtic, leptokurtic, or platykurtic, based on excess kurtosis scores. Mesokurtic refers to a normal distribution, with an excess kurtosis score of between -1 and 1. Leptokurtic and platykurtic, also known as light-tailed and heavy-tailed, are distributions with a negative and positive excess kurtosis score below and above -1 and 1, respectively. Platykurtic distribution are more prone to have outliers, found in their tails, than leptokurtic distributions [129], [130].

The parameter tailedness and symmetry were evaluated by categorizing their excess kurtosis and skewness scores, as seen in Table 6.4. The EVLP was typically platykurtic, with either a zero or right tailed asymmetry. The ASV measured parameters were mostly symmetric with all three types of tailedness. In general, the ASV and EVLP do not have matching tailedness except for the time parameters such as inspiratory time that are platykurtic. Only a few parameters had symmetry and tailedness of a normal distribution.

Table 6.5 Shapiro Wilk test of the case 1 ASV and EVLP measurements of the ventilator test lung

Parameter	ASV Normality	EVLP Normality	ASV p-Value	EVLP p-Value
Inspiratory Tidal Volume, *VTi* [mL]	×	×	1.33E-10	4.21E-05
Expiratory Tidal Volume, *VTe* [mL]	×	×	6.22E-04	9.00E-12
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	×	×	1.69E-10	8.66E-07
Expiratory Dynamic Compliance, C_{dyne} [mL/cm$H_2$0]	×	×	6.31E-04	3.79E-12
Inspiratory Time, *tI* [s]	×	×	1.04E-02	2.10E-04
Expiratory Time, *tE* [s]	×	×	1.85E-07	6.13E-07
Total Time, *tTot* [s]	×	×	3.19E-11	3.15E-08
Inspiratory Total Time Ratio, *tITot*	×	×	2.51E-03	2.12E-04
Inspiratory Expiratory Time Ratio, *tIE*	×	×	5.49E-04	1.99E-05
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	×	×	2.29E-05	4.23E-10
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	×	×	2.69E-11	2.31E-04
Respiratory Rate, *RR* [bpm]	×	×	7.29E-12	7.63E-09

The distribution normalities of all parameters were check using the Shapiro-Wilk normality test. The normality and p-values of these test for case 1 are summarized in Table 6.5. This is somewhat unexpected, considering some parameters had excess kurtosis and skewness scores within -1 and 1, indicating a normal distribution.

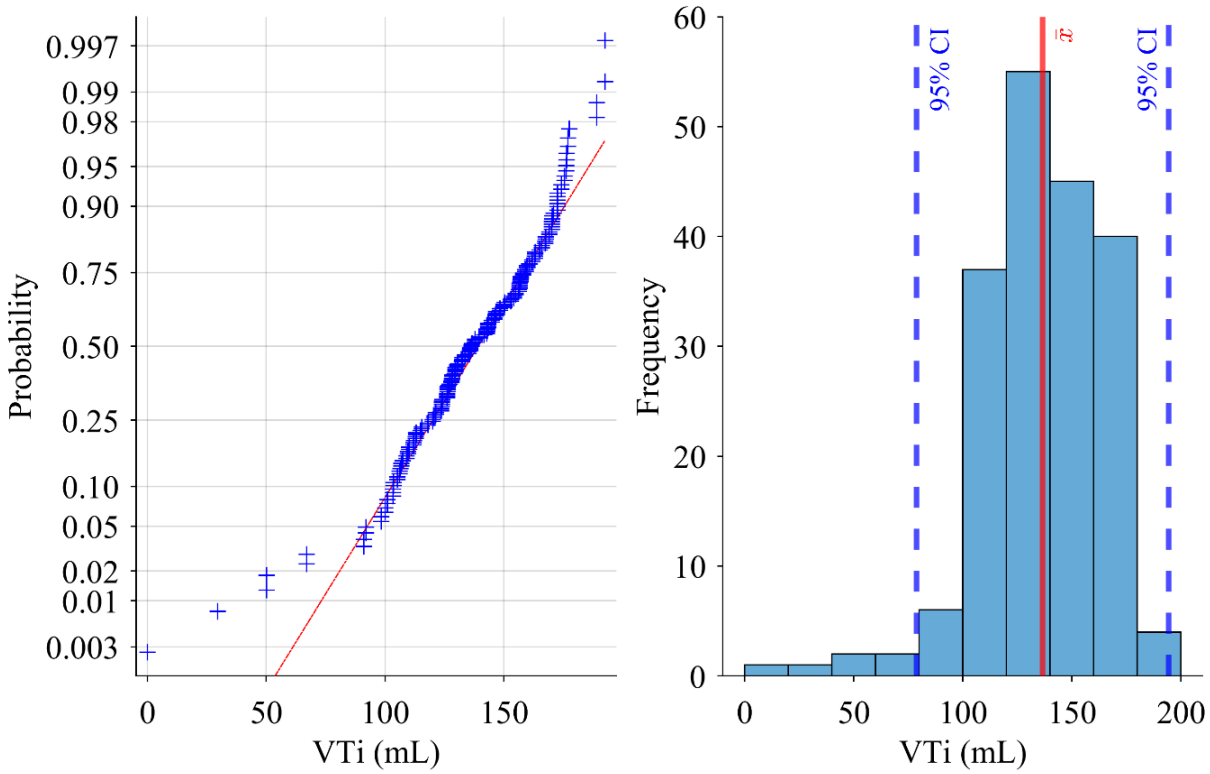


Figure 6.37 Plots of the ventilator test lung EVLP inspiratory tidal volume measurements from case 1, (a) normal probability and (b) histogram annotated with the mean = 136.52 and 95% confidence intervals [88.52,184.52]

The ASV and EVLP measurements were plotted to observe their measurement distributions. The EVLP inspiratory tidal volume measurements from case 1 were visualized using a probability plot, Figure 6.37 (a), and a histogram, Figure 6.37 (b). The probability shows the shape of the distribution relative to a standard normal distribution, the red line. The histograms are annotated with the mean, the red vertical line, and their 95% confidence intervals, the two dashed blue lines. These plots show that the EVLP case 1 tidal volume measurements have a near normal distribution. The probability plot follows the red normal distribution line, except below 5% and above 90% probability. Also, the histogram shows that the distribution is centered on the mean with a bell curve shape, perhaps with low tailedness. Lastly, both indicate that the measurements below 5% probability are outliers. The ASV measurements from case 1 were examined in the same way.

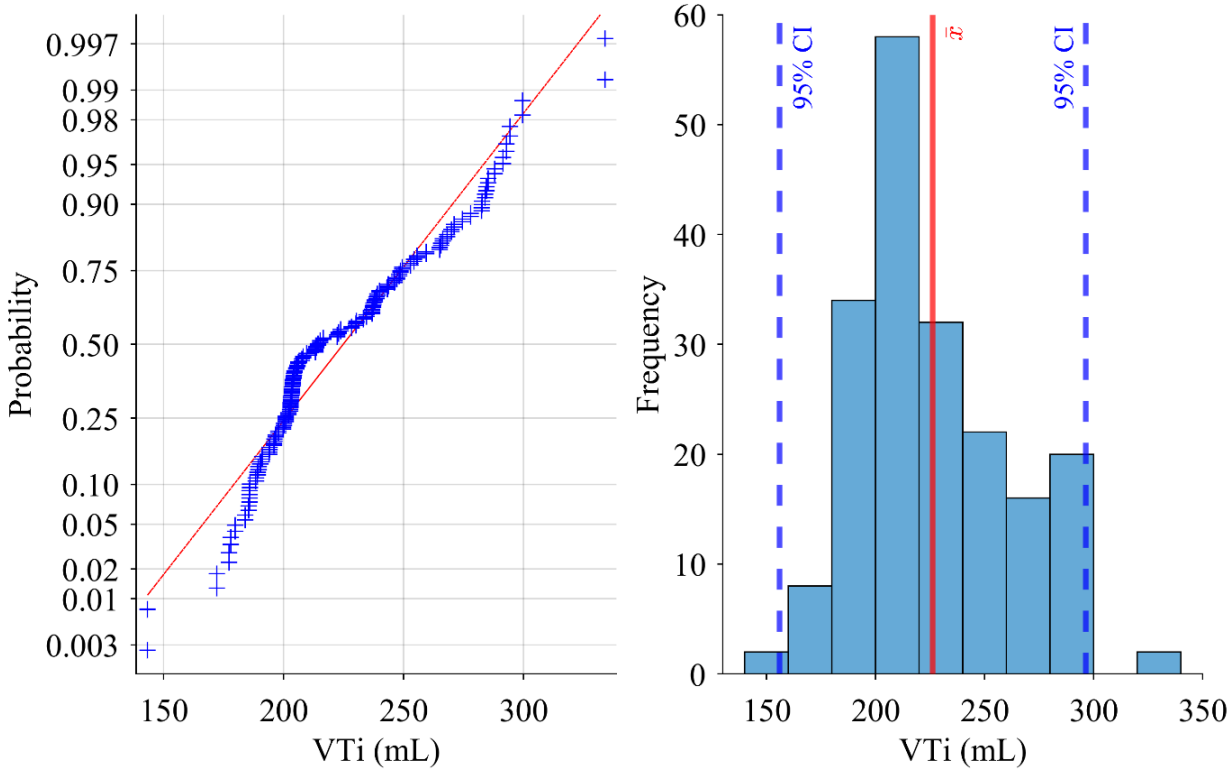


Figure 6.38 Plots of the ventilator test lung ASV inspiratory tidal volume measurements from case 1, (a) probability and (b) histogram annotated with the mean = 226.05, red line, and 95% confidence interval [155.67,296.43], blue dashed lines

The ASV tidal volume measurements from case 1 are less consistent and normal in shape than the EVLP. The probability plot does not closely follow normality, as seen in Figure 6.38 (a). Also, the histogram does not have a bell curve shape. Instead, it is flat with a peak near the mean, as seen in Figure 6.38 (b). These features suggest the distribution is left-tailed but have less outliers than the EVLP.

In general, the ASV and EVLP measurements do not have the same distribution shape or properties as demonstrated by comparing the case 1 tidal volume distributions. Specifically, they have significantly different means and the EVLP seems to have more repeatability because its confidence interval is narrower, and its shape is taller than wide. These results match the excess kurtosis and skewness scores. Although, the measurement distribution behavior varies between parameters and methods, most are centered on their means.

In summary, the ASV and EVLP measurement distributions are non-normal with significant mean error with varying distribution symmetry and tailedness. Also, the EVLP has a smaller standard deviation than the ASV. Lastly, the low mean error of the timing parameters suggests that the ASV is measuring physical displacement of the ventilator test lung along with the EVLP ventilation.

6.4.3 Correlation of the Ventilator Test Lung ASV and EVLP Measurements

The ASV and EVLP measurement correlation was measured to determine if the ASV measurements are dependent on the ventilator test lung displacement. If the ASV and EVLP are measuring the lung displacement, they are expected to have good correlation, and ideally a linear relationship. Therefore, correlation was measured using the Pearson correlation coefficient [132], and an adapted Kolmogorov-Smirnov hypothesis test was used to measure linearity [133]–[135]. Also, Passing-Bablok regression was used to create linear models for each tidal parameter using the ASV and EVLP paired measurements. The method makes no assumptions about the data's distribution, allows both sets of data to have measurement error, and is robust against outliers [135]. However, the Passing-Bablok regression is only applicable for continuous variables and assumes the data has a linear relationship [135]. Assuming that the model fits the sample data well, the slope and intercept of the model can be used to estimate the proportional and systematic bias between the systems [136], [137].

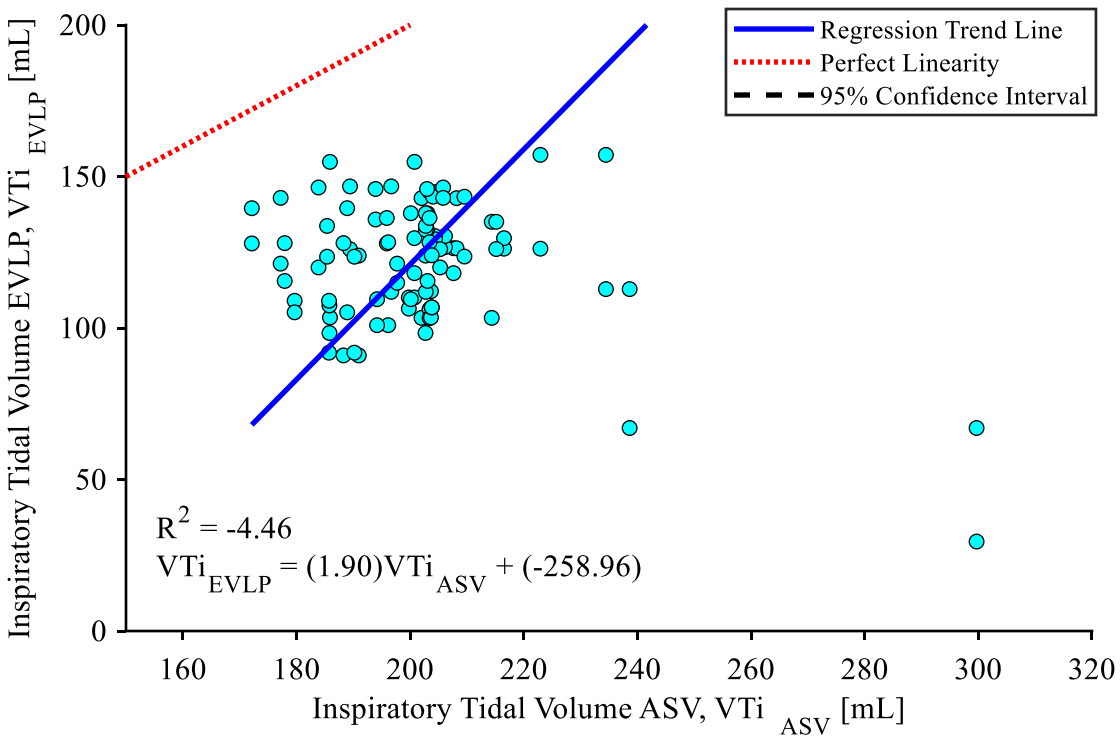


Figure 6.39 Plot of Passing-Bablok regression of the case 1 ASV and EVLP inspiratory tidal volume of the ventilator test lung

The experiment cases were individually measured for correlation and linearity, then modelled using Passing-Bablok regression. However, when the models and data were plotted, they were found to cluster around a single value. This tendency can be seen in the case 1 regression plot, as seen in Figure 6.39. Also, the measurements show more variance along the ASV axis. This trend occurs cases are when modeled individually for all parameters.

It is hypothesized that this distribution occurs because each case is an experimental configuration of two variables: the Intel RealSense D435 height, and the EVLP ventilation blower rate, which indirectly controls the tidal volume and airway pressure. Tidal volume, pressure, and dependent metrics would have target values that remain constant within cases but would change between cases. For all other parameters, their ventilation setting did not change between or within any experiment cases. If this is the case, then the regression figures and models are centered on these ventilation targets.

The Passing-Bablok regression method assumes that the samples are from a continuous distribution. Since the paired samples centre on the ventilation target values, this is not

achievable with the data from any case individually. As a result, the method and data are not expected to provide a strong correlation between the paired samples, and their population.

To mitigate this issue, regression models for each parameter were created using the paired samples from multiple cases. This step would not address the problem for any of the parameters with constant ventilation target values between cases. However, this approach will create multiple clusters for the tidal volume. This step will broaden the range of measurements, making it closer to a continuous dataset.

Accumulating data from multiple case will bias the regression model, as it will likely form a linear relationship that passes through the cluster centers. Likewise, the constant parameters will remain biased to having no significant relationship.

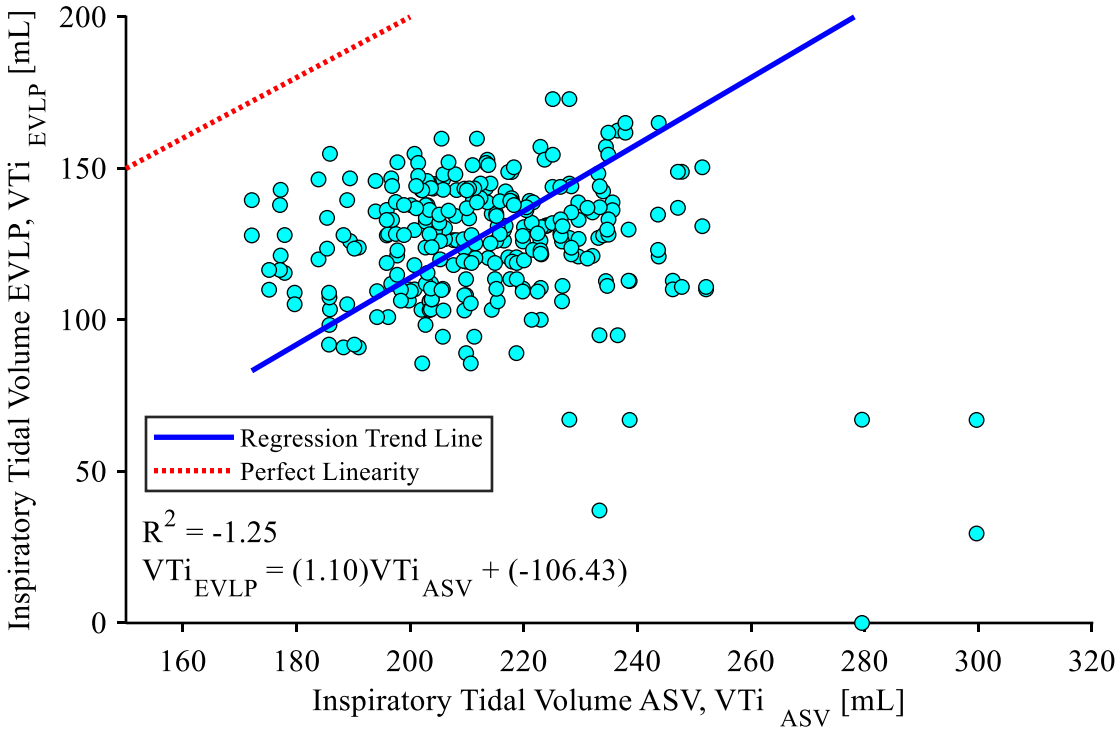


Figure 6.40 Plot of Passing-Bablok regression of the EVLP test lung inspiratory tidal volume from cases 1, 4, and 7

Six regression models were obtained for the groups of cases with the same experimental configurations levels, three for each level of the blower rate and three for each level of Intel RealSense D435 height. As see in Figure 6.40, the inspiratory tidal volume paired samples from

cases 1, 4, and 7 cluster around one point. Each case was taken at the same tidal volume but different Intel RealSense D435 heights. This may indicate that the Intel RealSense D435 height did not have a significant effect on the distribution of the measurements. This behavior was found in the combined regression models and other parameters. As a result, all nine experiment cases were measured as one combined dataset.

Table 6.6 Linearity of all experiment case ASV and EVLP measurements of the ventilator test lung

Parameter	Linearity Significant	Maximum Cumulative Sum Rank Difference	Critical Value
Inspiratory Tidal Volume, *VTi* [mL]	×	69.00	36.34
Expiratory Tidal Volume, *VTe* [mL]	×	71.00	36.34
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	×	43.33	36.11
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH20]	×	79.37	36.16
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	×	58.00	25.66
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	×	47.00	25.52
Inspiratory Time, *tI* [s]	×	36.00	25.66
Expiratory Time, *tE* [s]	×	26.00	25.44
Total Time, *tTot* [s]	×	44.00	25.37
Inspiratory Total Time Ratio, *tITot*	×	40.00	25.37
Inspiratory Expiratory Time Ratio, *tIE*	×	40.00	25.37
Respiratory Rate, *RR* [bpm]	×	50.00	25.37

The ASV and EVLP paired measurements were found to be non-linear for all parameters, as seen in Table 6.6. As previously mentioned, linearity and correlation are prerequisites for Passing-Bablok regression [135] and Pearson correlation is valid for measuring linear relationships [132]. Therefore, it is unlikely that the Passing-Bablok regression models and Pearson correlation coefficient will be valid.

Table 6.7 Correlation of all experiment cases ASV and EVLP measurements of the ventilator test lung

Parameter	Pearson Correlation Coefficient	Correlation Significant	Correlation p-value
Inspiratory Tidal Volume, *VTi* [mL]	0.3918	✓	1.31E-27
Expiratory Tidal Volume, *VTe* [mL]	0.3727	✓	6.08E-25
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	-0.0534	✓	1.57E-01
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH20]	0.1736	✓	3.42E-06
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	-0.0087	✓	8.70E-01
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	0.0332	✓	5.34E-01
Inspiratory Time, *tI* [s]	0.1310	✓	1.33E-02
Expiratory Time, *tE* [s]	0.1727	✓	1.18E-03
Total Time, *tTot* [s]	0.3430	✓	4.55E-11
Inspiratory Total Time Ratio, *tITot*	0.1376	×	0.27
Inspiratory Expiratory Time Ratio, *tIE*	0.1229	×	0.26
Respiratory Rate, *RR* [bpm]	0.4042	×	1.03

The Pearson correlation coefficient was measured between the ASV and EVLP paired measurements for all parameters and cases. For all data sets, most parameters were found to represent the population correlation, as seen in Table 6.7. It is hypothesized that the methods have poor correlation because of the significant difference in measurement variance, which was evaluated in Chapter 6.4.2. Despite the measurements having poor correlation and being non-linear, Passing-Bablok regression was used to further evaluate their relationship.

Table 6.8 Passing-Bablok regression parameters of the ASV and EVLP measurements from all experiment cases combined of the ventilator test lung

Parameter	Coefficient of Determination, R^2	Intercept	Slope	95% Confidence Interval			
				Slope		Intercept	
Inspiratory Tidal Volume, *VTi* [mL]	-0.02	18.40	0.50	0.46	0.54	7.62	28.09
Expiratory Tidal Volume, *VTe* [mL]	-0.02	0.14	0.62	0.56	0.67	-13.86	13.37
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	-1.86	-27.97	1.50	1.30	1.74	-35.21	-22.14
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	-1.41	-40.66	1.97	1.70	2.29	-50.30	-32.65
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	-25.76	-2.30	4.00	3.00	5.00	-3.00	-1.60
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	-0.03	1.23	0.05	0.03	0.08	1.19	1.27
Inspiratory Time, *tI* [s]	-0.45	1.72	0.29	0.22	0.37	1.52	1.88
Expiratory Time, *tE* [s]	-0.29	3.79	0.25	0.18	0.33	3.37	4.13
Total Time, *tTot* [s]	-1.63	-0.01	1.00	1.00	1.33	-2.49	0.00
Inspiratory Total Time Ratio, *tITot*	-0.47	0.24	0.27	0.21	0.35	0.21	0.26
Inspiratory Expiratory Time Ratio, *tIE*	-0.82	0.35	0.26	0.20	0.34	0.31	0.38
Respiratory Rate, *RR* [bpm]	-1.99	-0.22	1.03	1.00	1.29	-2.35	0.03

Passing-Bablok regression was used to model the linear relationship between the ASV and EVLP paired measurements for all tidal parameter and experiment cases. The slope, intercept, and coefficient of determination, R^2, was measured for each model. Also, the lower and upper 95% confidence intervals for the slope and intercept were found.

For case 1, the R^2 values indicates the data does not fit any of the models except for *PIP*, as seen in Table 6.8. These results match the correlation and linearity tests in Table 6.7. A slope of one and an intercept of zero are not within the confidence intervals of most parameters. Therefore, the 95% confidence intervals indicate that for most parameters, there is systematic and proportional bias. These biases are measured as the slope and intercept values. For all parameters, except the pressure metrics, the slope and intercept significantly deviate from one and zero and are substantial errors. The Passing-Bablok regression plots were examined to understand how and why the data is not linear and has a poor correlation.

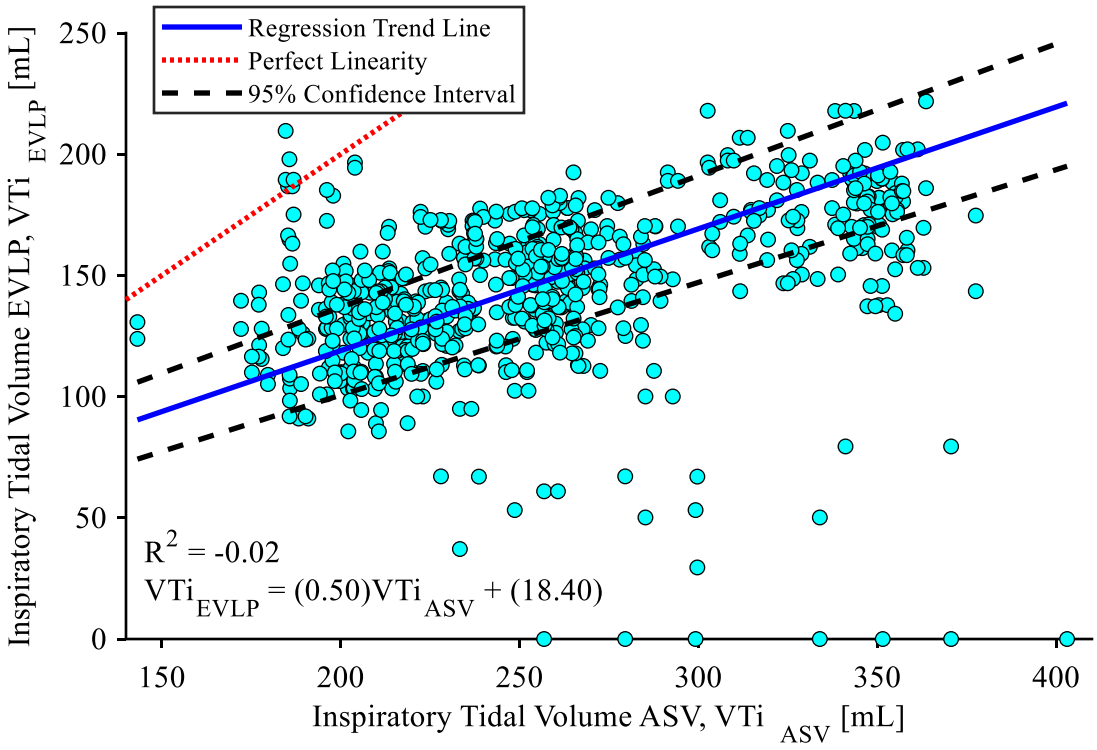


Figure 6.41 Plot of Passing-Bablok regression of the ASV and EVLP inspiratory tidal volume measurements of the ventilator test lung

The Passing-Bablok regression plots are scatter plots of the paired ASV and EVLP measurements of a tidal parameter for a specific case. They are annotated with a 95% confidence interval, the trend line, and the trend line of a perfectly linear relationship. The paired tidal volume measurements from all cases were regressed together, as seen in Figure 6.41. The plots show that combining datasets from different blower rate experiment configurations expanded the tidal volume dataset range, allowing the regression to model a linear trend.

Despite, the poor correlation and coefficient of determination, the regression model fits the data as it passes through the center of the measurement distribution. Also, the dataset is linear but with many outlier data points and significant dispersion. Lastly, this plot confirms that the ASV method is overestimating the EVLP by a significant amount. The dynamic compliance regression plot looks the same as the tidal volume plot. This is unexpected, since the compliance of the ventilator test lung property should be constant. However, this relationship suggests that the ventilator test lung became more compliant with higher tidal volumes, or over the course of multiple experiment cases. The other parameters did not have this trend though.

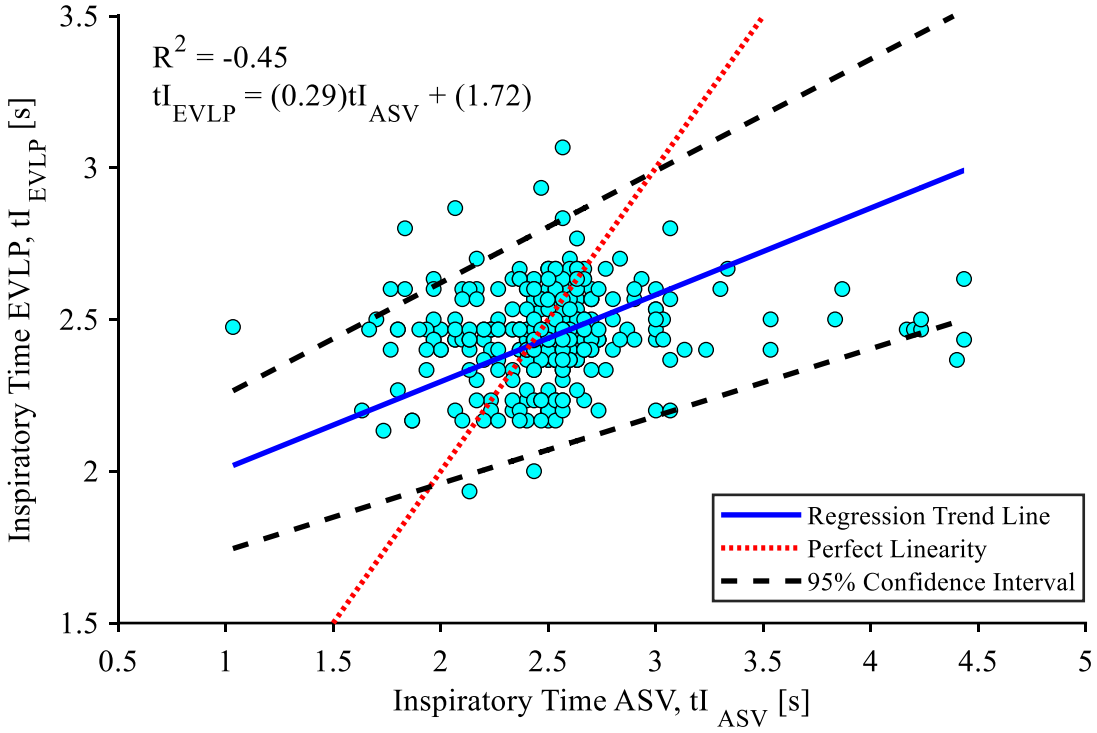


Figure 6.42 Plot of Passing-Bablok regression of the ASV and EVLP inspiratory time measurements from all cases of the ventilator test lung

The data points of most parameters form a single cluster, like in the regression plot of all measurements of inspiratory time from all experiment cases, as seen in Figure 6.42. Despite the data points clustering, the ASV and EVLP measurements form a linear relationship with poor correlation and coefficient of determination. This trend formed because of the ASV method has a larger variance than the EVLP. As a result, the data points are spread across the *x*-axis, almost forming a horizontal line.

Table 6.9 Residual mean of Passing-Bablok regression for all valid cases of the ventilator test lung

Parameter	Residual Mean	Normalized Residual Mean [%]
Inspiratory Tidal Volume, *VTi* [mL]	-2.59	-0.08
Expiratory Tidal Volume, *VTe* [mL]	-2.76	-0.07
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	-0.05	-0.01
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	0.43	0.06
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	-0.10	-0.21
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	0.05	0.14
Inspiratory Time, *tI* [s]	0.01	0.04
Expiratory Time, *tE* [s]	0.01	0.05
Total Time, *tTot* [s]	0.00	0.01
Inspiratory Total Time Ratio, *tITot*	0.01	0.04
Inspiratory Expiratory Time Ratio, *tIE*	0.00	-0.03
Respiratory Rate, *RR* [bpm]	-0.01	-0.04

The Passing-Bablok regression was followed by residual analysis to identify outliers and to evaluate if the regression provided an adequate fit for the sample set. Two types of residual plots, residual and standardized residual, were made for each parameter. The residual means for both types of plots for all cases when combined are summarized in Table 6.9 Residual mean of Passing-Bablok regression for all valid cases of the ventilator test lung. These residual means suggest that the regression models fit the data well since both types of means are low. The plots themselves were examined for any trends, specifically tidal volume since it is one of the only parameters that has a wider measurement range.

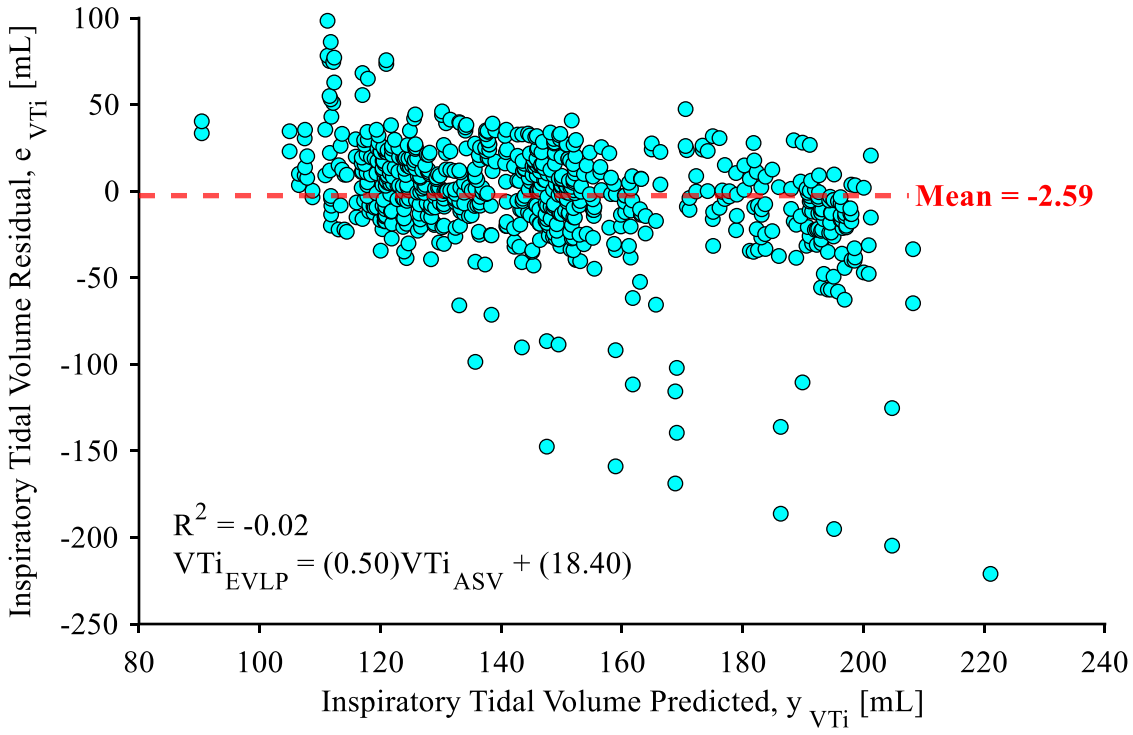


Figure 6.43 Plot of Passing-Bablok regression residuals versus predicted ventilator test lung inspiratory tidal volume from all experiment cases

The residual plots show the regression model trend lines pass through the center of the measurement distributions since they means are near zero. Also, they show few trends as most data points follow the horizontal residual mean line, as seen in Figure 6.43 for inspiratory tidal volume of all experiment cases. For other parameters, the residual plots are similar except the data points form a cluster centered on the residual mean. Also, the normalized residual plots show the same trends as the original residual plots, except the units are scaled by the distribution's standard deviation. Overall, these residual plots suggest the regression models fit well to the data, but the paired measurements have large dispersion and have many outliers causing poor model performance.

In summary, the ASV and EVLP paired measurements have poor correlation and do not have statistically significant linearity. This is supported by the coefficient of determination of the Passing-Bablok regression models. However, the regression and residual plots show that the regression passes through the center of the paired measurement distributions. As a result, the

regression fits well with the data, but performance is poor because of the large variance of the ASV measurements and several outlier points.

6.4.4 ASV and EVLP Measurement Agreement of the Ventilator Test Lung

Agreement was measured between the ASV and EVLP measurements, to determine if the ASV method was interchangeable with the EVLP system. Agreement is a measure of concordance between different assessments [132] that is often measured using intra-class correlation [132], [138]–[140] and Bland-Altman analysis [137], [140], [141]. The ASV and EVLP agreement was measured using all the experiment case data, for the same reasons as in Chapter 6.4.3.

Intra-class correlation (ICC) is a quantitative measure of agreement and correlation, which ranges between zero and one, where one indicates perfect agreement [132], [138]–[140]. There are several types based on the application [138], but the Absolute Case 2 type was selected for this application. Also, an F-test where the null hypothesis is that ICC is equal to zero, was performed on the paired samples to determine if the results are significant [138].

Bland-Altman analysis examines the distribution of the differences of two methods by plotting them against the mean of each pair of measurements [137], [140], [141]. The differences can be normalized to remove proportional bias. Also, the differences can be plotted against the reference method's measurements, which in this case is the EVLP. However, it is considered controversial as this imposes a relationship between the differences and magnitude [137].

Bland-Altman assumes that the differences of the two methods are normally distributed, and a linear relationship is present. Also, the method is only applicable for continuous variables. It is useful for identifying a systematic bias between the methods based on their mean difference. Also, different relationships in the two methods can be identified by visualizing the differences [137], [140], [141]. Lastly, limits of agreement can be interpreted by medical professionals to determine if it is acceptable to interchange methods [137].

Table 6.10 Intra-Class Correlation of the ventilator test lung ASV and EVLP measurements of all valid cases

Parameter	Intra-Class Correlation	Significant ICC	p-value
Inspiratory Tidal Volume, *VTi* [mL]	0.081	×	0.200
Expiratory Tidal Volume, *VTe* [mL]	0.110	×	0.176
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	-0.007	×	0.763
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH20]	0.030	×	0.210
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	-0.004	×	0.563
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	0.021	×	0.333
Inspiratory Time, *tI* [s]	0.080	×	0.061
Expiratory Time, *tE* [s]	0.103	✓	0.024
Total Time, *tTot* [s]	0.303	✓	0.000
Inspiratory Total Time Ratio, *tITot*	0.079	×	0.065
Inspiratory Expiratory Time Ratio, *tIE*	0.057	×	0.136
Respiratory Rate, *RR* [bpm]	0.341	✓	0.000

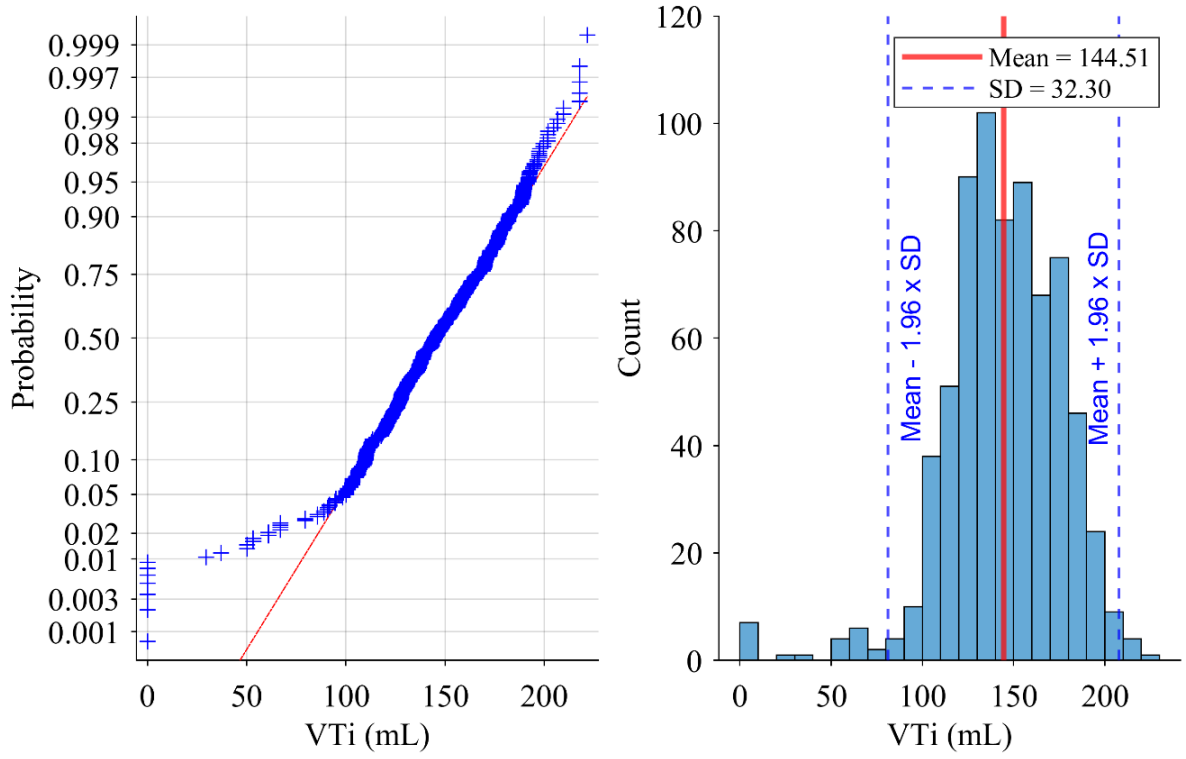
The ICC and F-test results for paired samples from all experiment cases are summarized in Table 6.10. All parameters were found to have poor ICC. Also, only three parameters were found to have non-zero ICC based on the F-test. Overall, these results indicate the ASV and EVLP do not agree. Agreement was examined using the Bland-Altman method.

Table 6.11 Shapiro-Wilk normality of the ventilator test lung ASV and EVLP measurement differences of all valid cases

Parameter	Method Difference Normality	p-value
Inspiratory Tidal Volume, *VTi* [mL]	×	5.55E-16
Expiratory Tidal Volume, *VTe* [mL]	×	2.13E-13
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	×	0.00E+00
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	×	0.00E+00
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	×	0.00E+00
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	×	1.41E-06
Inspiratory Time, *tI [s]*	×	7.77E-16
Expiratory Time, *tE [s]*	×	6.44E-15
Total Time, *tTot [s]*	×	0.00E+00
Inspiratory Total Time Ratio, *tITot*	×	1.11E-15
Inspiratory Expiratory Time Ratio, *tIE*	×	0.00E+00
Respiratory Rate, *RR* [bpm]	×	0.00E+00

A prerequisite for Bland-Altman analysis is that the differences have a normal distribution. The normality of the difference of the ASV and EVLP measurements for all tidal parameters were found using the Shapiro-Wilk test, as seen in Table 6.11. All parameters were found to have a

non-normal distribution. This invalidates the Bland-Altman limits of agreement, as they are computed with this assumption.



(a) *(b)*

Figure 6.44 Plots of the difference of the ASV and EVLP inspiratory tidal volume measurements from all experiment cases as (a) probability and as (b) histogram annotated with the mean and confidence intervals

The differences were visualised using probability and histogram plots to identify how the distributions deviated from normality. The inspiratory tidal volume difference plots show that the distribution looks normal, since it follows a normal probability, as seen in Figure 6.44 (a). Also, the histograms have a bell curve shape, as seen in Figure 6.44 (b). Therefore, the limits of agreement may not be valid.

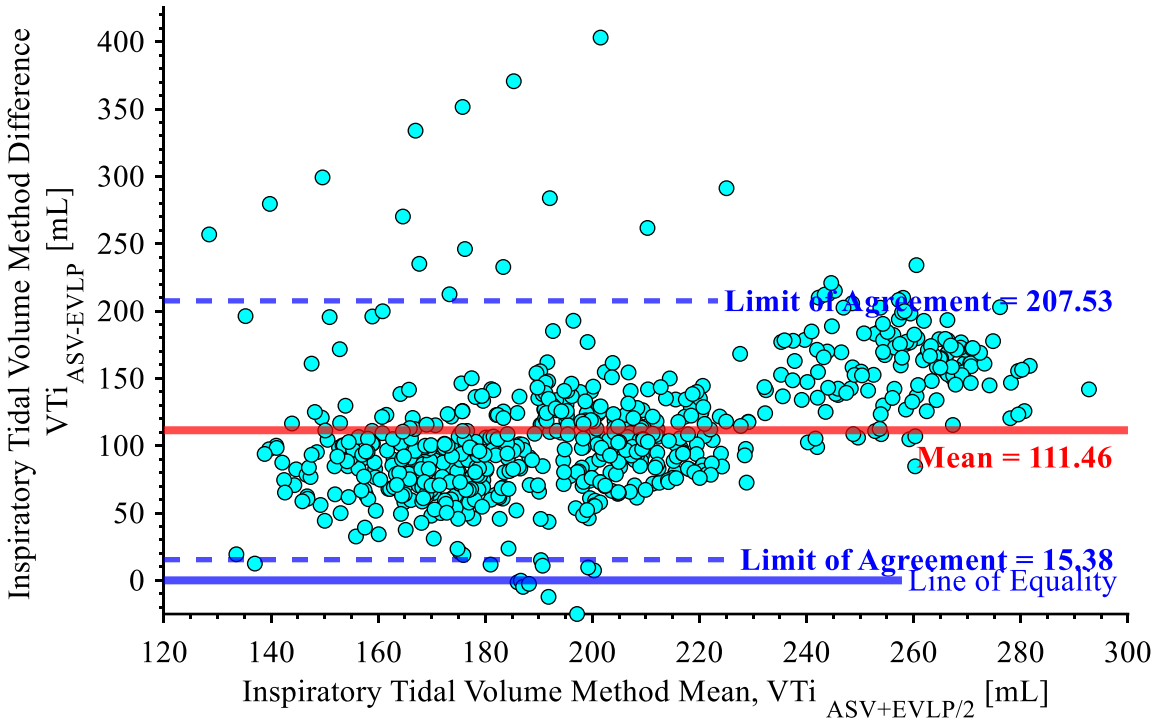


Figure 6.45 Plot of the Bland-Altman analysis of the ASV and EVLP measurements of inspiratory tidal volume of the ventilator test lung

For the reasons described in Chapter 6.4.3, the Bland-Altman analysis was performed on the paired measurements from all experiment cases combined. The mean of the differences is shown as a horizontal red line, along with the lower and upper limits of agreement as blue lines. The limits of agreement are found as the 95% confidence intervals of a normal distribution, calculated from standard deviation [137], [141].

The Bland-Altman plots for most of the parameters formed a single cluster, like in the regression plots in Chapter 6.4.3. The inspiratory tidal volume and dynamic compliance Bland-Altman plot were the exception. The data points clustered around three points in the Bland-Altman plot in Figure 6.45, just as it was seen in Chapter 6.4.3. It is hypothesized these three clusters are centered on the tidal volume achieved from the three blower rate settings in the different experiment configurations. Also, the differences slightly decrease with respect to the mean of the paired measurements, indicating proportional bias. The mean of the differences can be used as an estimate of the systematic bias between paired measurements. Outliers can be identified as being outside the limits of agreement.

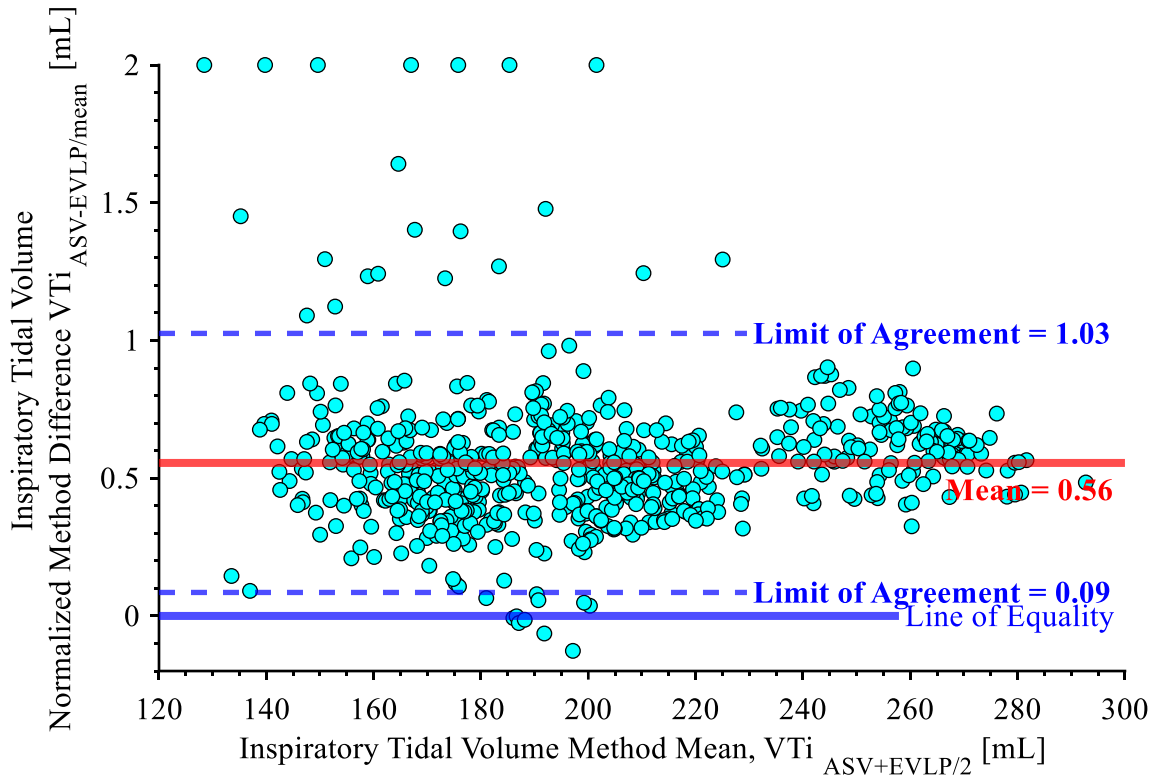


Figure 6.46 Plot of the Bland-Altman analysis of the ASV and EVLP normalized measurements of inspiratory tidal volume of the ventilator test lung

The inspiratory tidal volume differences were normalized and plotted, as seen in Figure 6.46. Also, the mean and limits of agreement are normalized. In this case, Figure 6.46 shows the same behaviour as Figure 6.45 without the proportional bias. Another difference is that the normalized plot shows the outliers more clearly below the limits of agreement.

Table 6.12 Bland-Altman analysis mean, confidence interval, and limits of agreement of the ASV and EVLP measurements from all valid cases for the ventilator test lung

Parameter	Mean Difference	Mean Difference 95% Confidence Interval		Limit of Agreement	
Inspiratory Tidal Volume, *VTi* [mL]	111.46	108.43	114.48	15.38	207.53
Expiratory Tidal Volume, *VTe* [mL]	100.85	97.60	104.10	-2.41	204.10
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH20]	13.07	12.75	13.38	3.18	22.95
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH20]	11.40	11.10	11.70	1.78	21.02
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	0.16	0.15	0.17	-0.12	0.44
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	0.27	0.19	0.36	-1.56	2.11
Inspiratory Time, *tI* [s]	0.07	0.04	0.11	-0.71	0.86
Expiratory Time, *tE* [s]	-0.07	-0.11	-0.04	-0.88	0.73
Total Time, *tTot* [s]	-0.01	-0.02	0.01	-0.35	0.33
Inspiratory Total Time Ratio, *tITot*	0.01	0.01	0.01	-0.09	0.11
Inspiratory Expiratory Time Ratio, *tIE*	0.03	0.02	0.05	-0.26	0.33
Respiratory Rate, *RR* [bpm]	0.01	-0.01	0.03	-0.38	0.40

The Bland-Altman analysis results are summarized in Table 6.12. The results include the mean of the residual, the confidence intervals for the mean of the residual, and the limits of agreement. These results validate the ICC measurements, indicating that the ASV and EVLP have poor agreement. Ideally, the mean of the residual is zero, indicating no systematic bias. As seen in Table 6.12, most of the tidal volume and flow rate derived metrics have significant residual means, indicating systematic bias. For example, the inspiratory tidal volume difference mean is 111.46. Also, systematic bias is present because the confidence intervals do not include zero. Furthermore, the limits of agreement indicate poor agreement because they are very wide. For reference, the EVLP sensor has a flow rate total error band of ± (2.5 + 0.5 FS) % while the ASV inspiratory tidal volume limits of agreement are ± 96.14. It is very unlikely this performance would be an acceptable substitute for clinical application. The timing and pressure metrics have low residual means, as seen in Table 6.12. However, the timing residual means might be significant for the scale of its measurements. The residual means were normalized to determine they have significant systematic bias relative to the scale of the measurements.

Table 6.13 Bland-Altman analysis normalized mean and limits of agreement of the ASV and EVLP measurements from all valid cases of the ventilator test lung

Parameter	Normalized Mean Difference	Normalized Limit of Agreement	
Inspiratory Tidal Volume, *VTi* [mL]	0.56	0.09	1.03
Expiratory Tidal Volume, *VTe* [mL]	0.50	-0.04	1.04
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	0.57	0.03	1.11
Expiratory Dynamic Compliance, C_{dyne} [mL/cmH_20]	0.49	0.02	0.96
Time to Peak Tidal Inspiratory Flow, *tPTIF* [s]	0.24	-0.13	0.61
Time to Peak Tidal Expiratory Flow, *tPTEF* [s]	0.05	-1.06	1.17
Inspiratory Time, *tI* [s]	0.02	-0.28	0.31
Expiratory Time, *tE* [s]	-0.02	-0.20	0.16
Total Time, *tTot* [s]	0.00	-0.05	0.05
Inspiratory Total Time Ratio, *tITot*	0.02	-0.27	0.31
Inspiratory Expiratory Time Ratio, *tIE*	0.04	-0.41	0.48
Respiratory Rate, *RR* [bpm]	0.00	-0.05	0.05

The normalized results are a percentage relative to the mean of the paired samples, giving measures of performance that are independent of the order of magnitude. The normalized Bland-Altman analysis results are summarized in Table 6.13. The confidence interval was not normalized because the normal confidence intervals already confirmed the presence of systematic bias. The normalized residual means validate that there is significant systematic bias. Tidal volume derived metrics were found to have normalized residual means around 50%. However, the timing derived metrics were found to have low normalized residual means that indicate around a 5% error. Lastly, the normalized limits of agreement indicates poor agreement, supporting previous measurements.

In summary, the ASV and EVLP methods have poor agreement based on intra-class correlation measurements and Bland-Altman analysis. The tidal volume derived metrics were found to have large systematic error based on the confidence intervals of residual means, the residual means, and limits of agreement. However, the timing metrics were found to sometimes have reasonable agreement, high ICC, and low residual means.

6.5 Discussion

6.5.1 Sources of Error

The Bland Altman and Passing-Bablok analysis found significant systematic and proportional bias in the tidal volume measurements between the ASV and EVLP methods. However, the ASV and EVLP methods had reasonable agreement in measuring respiratory cycle timing parameters such as respiratory rate and expiratory time. Furthermore, the ASV method has a slightly smaller measurement standard deviation than the EVLP for tidal volume and dynamic compliance. This implies that the ASV method was able to consistently measure the ventilator test lung's displacement, especially the timing, but with an inaccurate intensity.

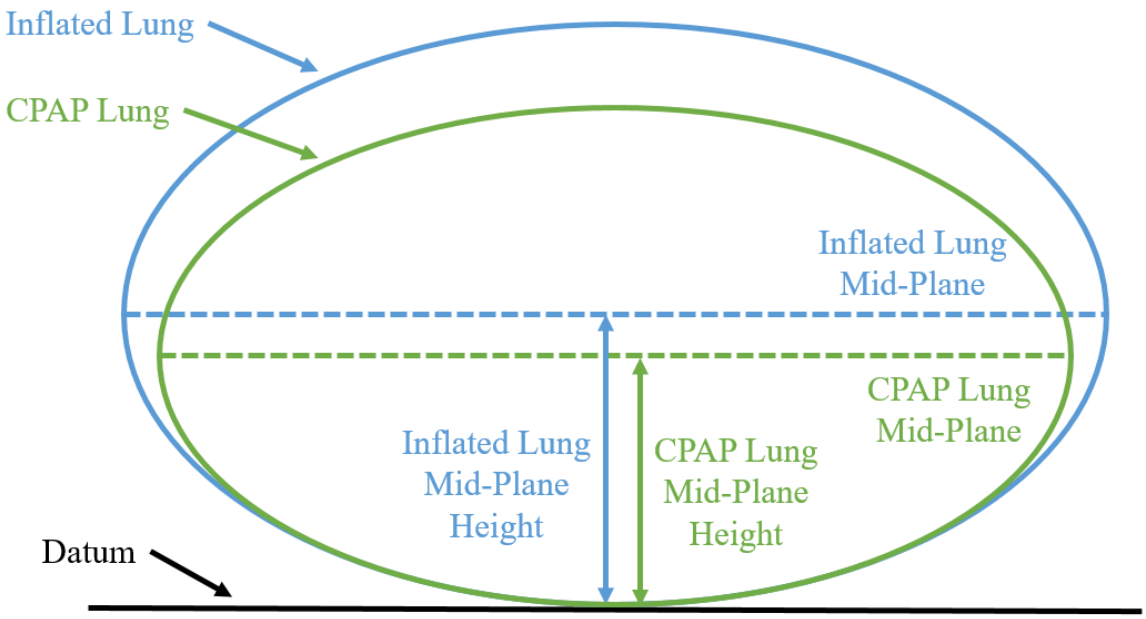


Figure 6.47 Schematic of the ventilator test lung distending in all directions, lifting the entire lung

It is hypothesized when the ventilator test lung inflates in all directions, it lifts the entire lung including its top surface. Therefore, instead of the ASV method just measuring the displacement of the top surface, it is measuring this systematic and proportional offset in surface height causing the tidal volume measurements to increase. In this case, the distance between the center of the ventilator test lung and the datum would increase during ventilation, as seen in Figure 6.47. This would lift the entire ventilator test lung, in addition to the displacement of the top surface, increasing the measured displacement using the surface integration method.

The curved surface of the ventilator test lung might have caused the occlusion of one or both infrared ASVs in the Intel RealSense D435. As a result, depth holes formed along the perimeter of the ventilator test lung. This effect might be more prominent at higher tidal volumes because the surface curvature would increase with displacement. Therefore, this occlusion could have systematically and proportionally decreased the estimated volume of the ventilator test lung. This effect would impact the tidal volume and dynamic compliance.

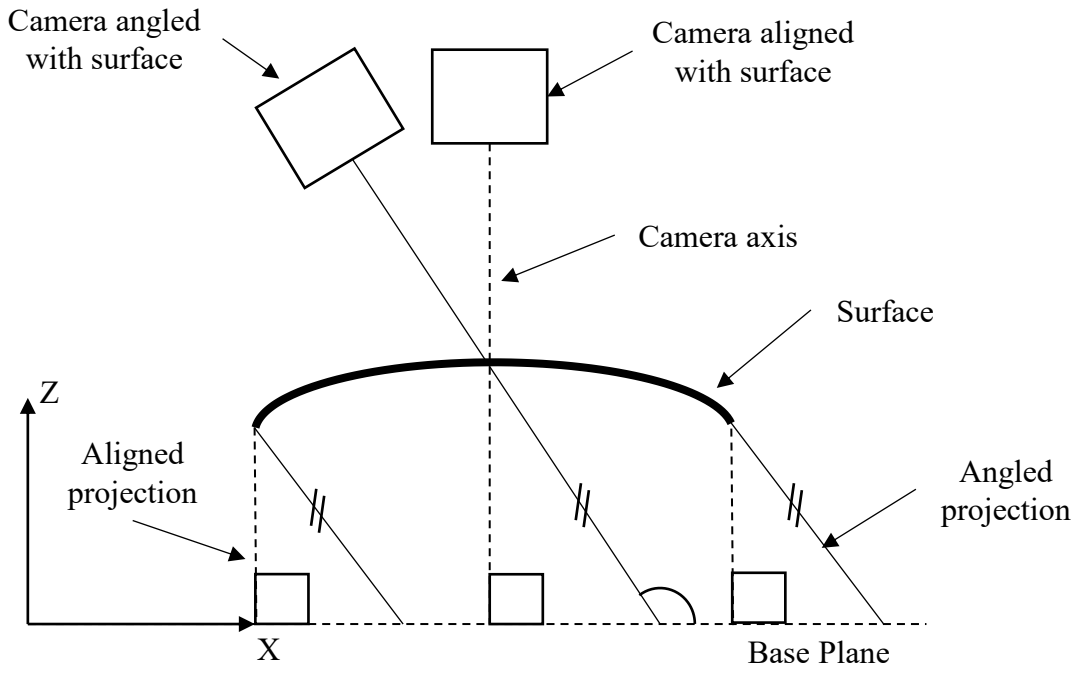


Figure 6.48. Schematic of the effect of ASV angle on surface integration

The Intel RealSense D435 perspective could have caused another issue. The surface integration method could have been impacted by any misalignment between the EVLP and Intel RealSense D435. Any angular deviation from perpendicularity between the ventilator test lung and Intel RealSense D435 would cause the integration method to estimate a different volume, as seen in Figure 6.48. This is an issue because the ventilator test lung and Intel RealSense D435 are assumed to be perpendicular. This assumption is used to transform the surface point cloud from the Intel RealSense D435 perspective into another more favorable coordinate system.

6.5.2 **ASV Method Limitations**

The main limitation of the ASV method is that it does not track the surface of the ventilator test lung, as described in Chapter 6.3.4. The regional measurements are distorted by the ventilator test lung as it moves to a different position throughout the respiratory cycle. As a result, the surface tidal displacement and surface dynamic compliance do not represent the displacement of the lung across its surface. Also, the regional measurements cannot be used for peak detection because the ventilator test lung's movement enforces maxima and minima regions that are correlated to localized over-inflation.

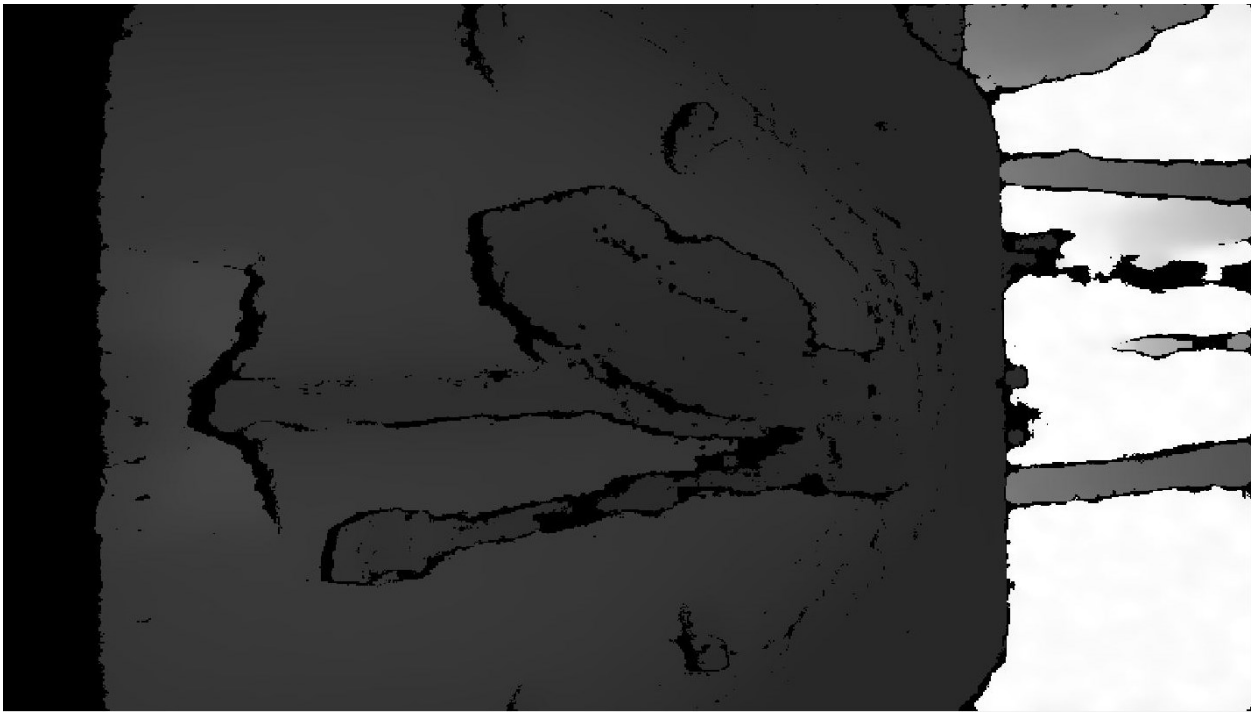


Figure 6.49 Image of the depth map of the ventilator test lung with depth holes that outline the lung

The edge-based segmentation method was another limitation of the ASV method in this experiment. Specifically, the segmentation method was found to be dependent on the depth holes caused by occlusion to create a valid edge map to segment the ventilator test lung, as seen in Figure 6.49. The occlusion created depth holes that outline the ventilator test lung and other objects.

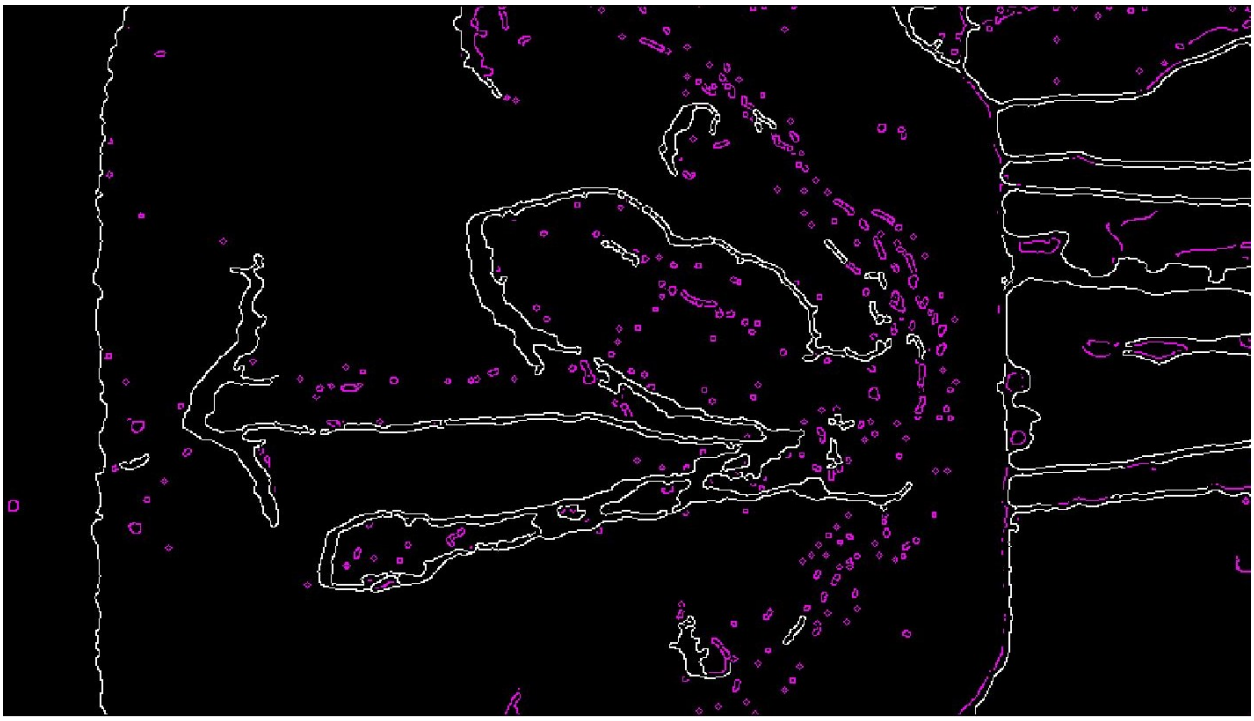


Figure 6.50 Image of the ventilator test lung edge map with circles from small regions of occlusion in the depth map

The Canny edge detection method only returned edges along the perimeter of depth holes, as seen in Figure 6.50. It is hypothesized it only returned depth hole edges because they were the only “strong” edges found using hysteresis thresholding. This could be due to the occlusion edges being significantly greater than the other edges, raising the parametric upper and lower thresholds above the other edges. As a result, the edge map of the ventilator test lung has many circles formed by the depth holes seen in Figure 6.49. The isolated circles that were removed by the processing scheme are colored in magenta. Also, dependence on these edges makes the segmentation method vulnerable to stray objects in the scene, and limits which edges in the depth map will be used for edge-linking.

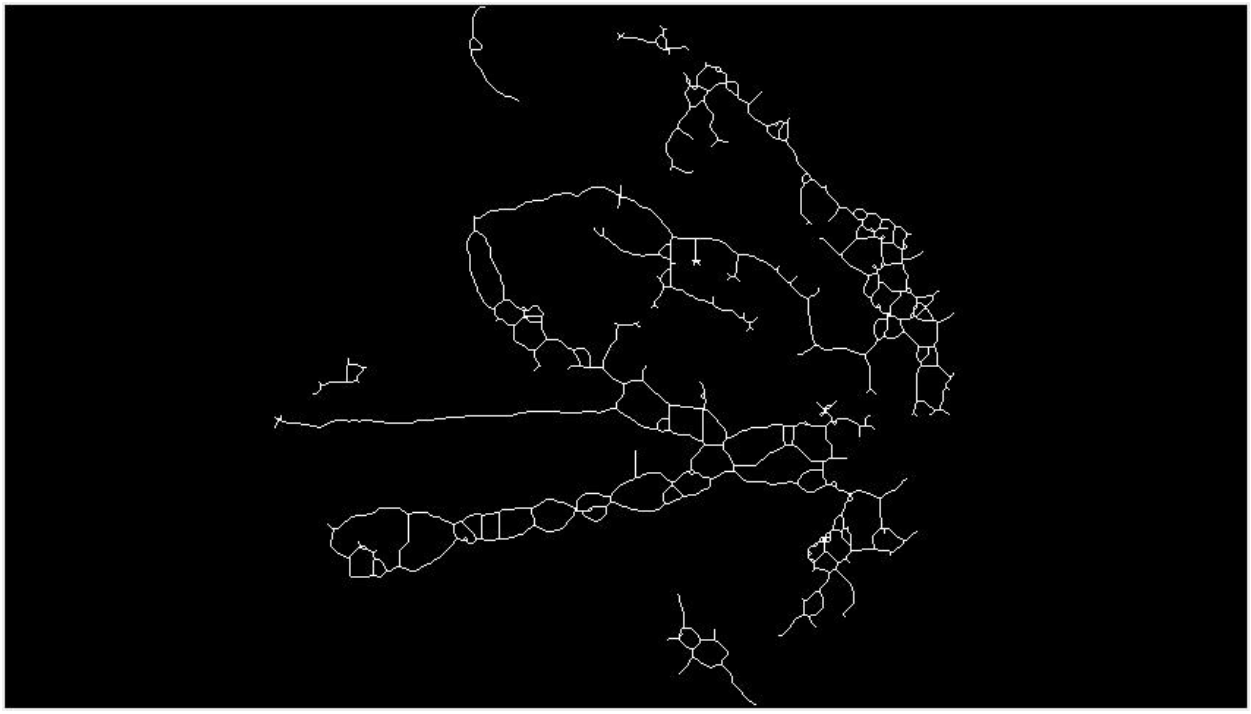


Figure 6.51 Image of the edge map of the ventilator test lung with erroneous spurs and sub-regions

During edge linking, the edges from the small regions of depth hole form sub-regions inside the lung, and create spurs along the lung's perimeter, as seen in Figure 6.51. This edge map is covered in erroneous spurs and sub-regions that will either cause segmentation failure or will lead to erroneous segmentation of the ventilator test lung. This segmentation error caused the experiment case 3 dataset to have an inaccurate displacement signal from surface integration, which resulted in invalid plethysmography measurements

The segmented region will not accurately represent the ventilator test lung. These problems could be mitigated with another segmentation method, that is less sensitive to occlusion, or by removing the depth holes in the depth maps. However, other segmentation approaches were attempted, such as depth thresholding, were found to be difficult to implement.

6.6 Conclusion

The surface displacement of a ventilator test lung was measured by the Intel RealSense D435 during mechanical ventilation performed by an EVLP. The Intel RealSense D435 measurements were used to obtain plethysmography and regional measurements using the processing scheme described in Chapter 5. Also, the EVLP derived the same plethysmography measurements from its flow rate and pressure measurements. The ASV measurement system was evaluated by comparing its measurements with the EVLP method.

The individual experiment case datasets were found to cluster around a single point that was dependent on ventilation tidal volume. Analyzing an individual experiment case led to invalid results because they do not have a wide measurement range. Therefore, all samples from all experiment cases were combined into one data to compare the methods.

Correlation was measured using Pearson correlation coefficient and Passing-Bablok regression. The ASV and EVLP methods were found to have poor a Pearson correlation coefficient. The regression models' confidence intervals, slope, and intercept indicate the methods have systematic and proportional error. Also, the regression models were found to have a poor coefficient of determination despite the models passing through the center of the measurement distribution. It was hypothesized that the models performed poorly because the ASV measurements have high dispersion.

Agreement was measured using intra-class correlation and Bland-Altman analysis. The ASV and EVLP methods were found to have poor intra-class correlation. Also, the residual means of the Bland-Altman plots were large for the tidal volume derived metrics, indicating poor agreement due to systematic error. However, the residual means were small for the timing metrics suggesting the ASV measured the same respiratory cycle changes as the EVLP.

7 Active Stereo Vision Method in a Clinical Setting

The study in Chapter 6 provides a general idea of the method's performance. However, the ventilator test lung and human lung are dissimilar in shape, scale, and most importantly compliance, as elastomer does not perfectly mimic tissue. Also, the EVLP was repurposed for the study, as it was not intended to perform positive pressure ventilation.

A better human lung surrogate are porcine lungs, which are commonly used in medical studies. A pair of porcine lungs were acquired, from another EVLP study, and ventilated with a clinical ventilator unit to perform another method comparison analysis. The porcine lung was measured with the Intel RealSense D435 and ventilator unit system. The data was processed for plethysmography measurements using the previously described processing scheme. However, a different image segmentation method was used than the method used in Chapter 6. Also, only a few plethysmography measurements were accessible from the ventilator unit, limiting the scope of the method comparison analysis.

7.1 **Experiment Equipment**

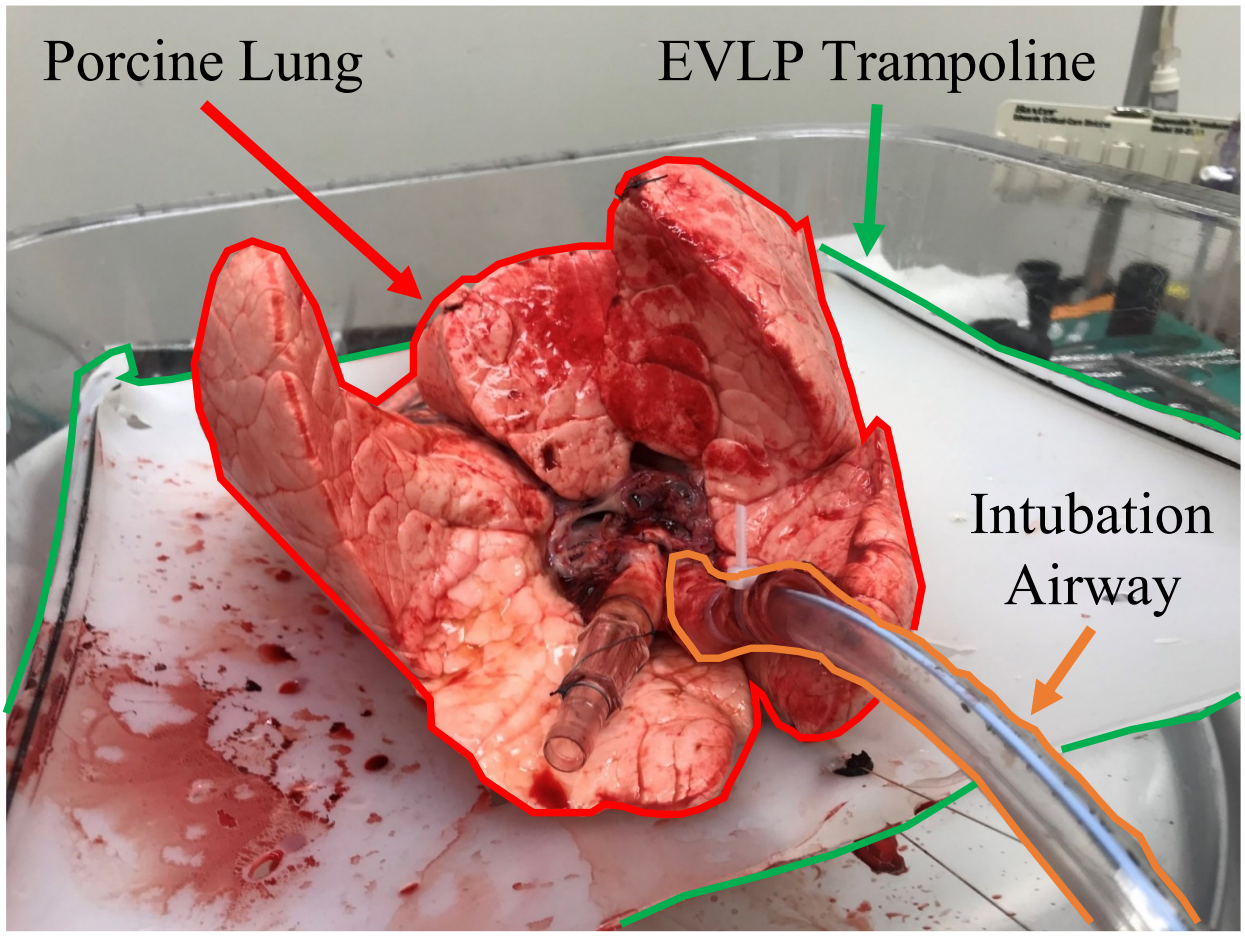


Figure 7.1 Annotated image of a porcine lung inside an open EVLP

The porcine lung, seen in Figure 7.1, was excised from a 75 kg donor for another EVLP study for 12 hours. As part of this previous experiment, the porcine lung was sealed within an EVLP, resting on top of the EVLP compliant trampoline seen in Figure 7.1. Also, the lung was intubated to form an airway connection to the EVLP. To access the trachea for intubation, the rough face of the porcine lung was faced upward, resting the smooth face directly on the trampoline.

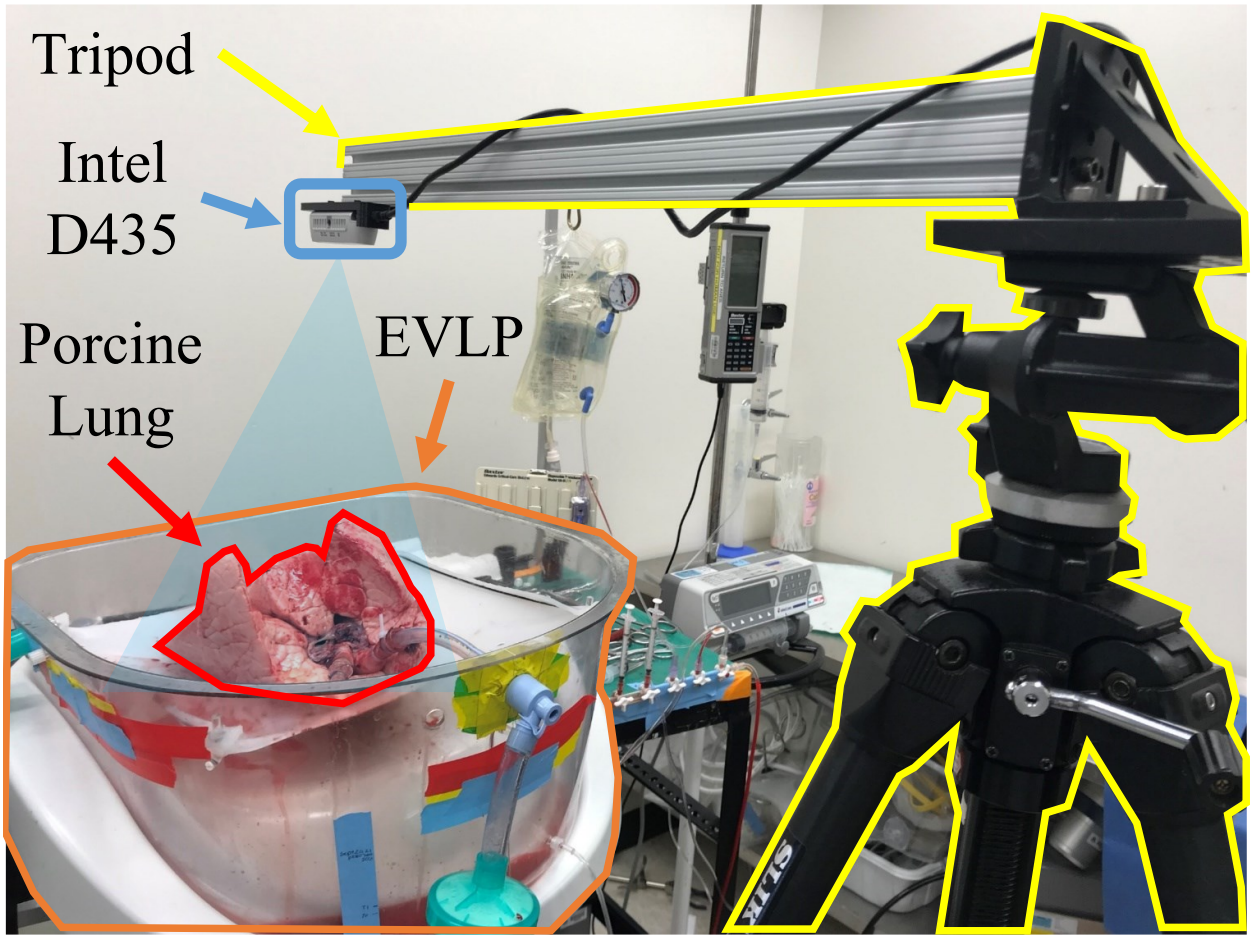


Figure 7.2 Annotated image of the porcine lung inside an EVLP with the Intel RealSense D435

The equipment and setup from the previous experiment was re-used for this study. Specifically, the porcine lung remained on top of the EVLP trampoline, and the intubated airway was reused for ventilation. Similarly, the same tripod for the Intel RealSense D435 from Chapter 6 was reused. However, some parts of the previous experiment setup were modified for this experiment. The EVLP cover was removed to give the Intel RealSense D435 a direct line of sight to the porcine lung from above, as seen in Figure 7.2. Also, the EVLP was not used to ventilate the porcine lung.

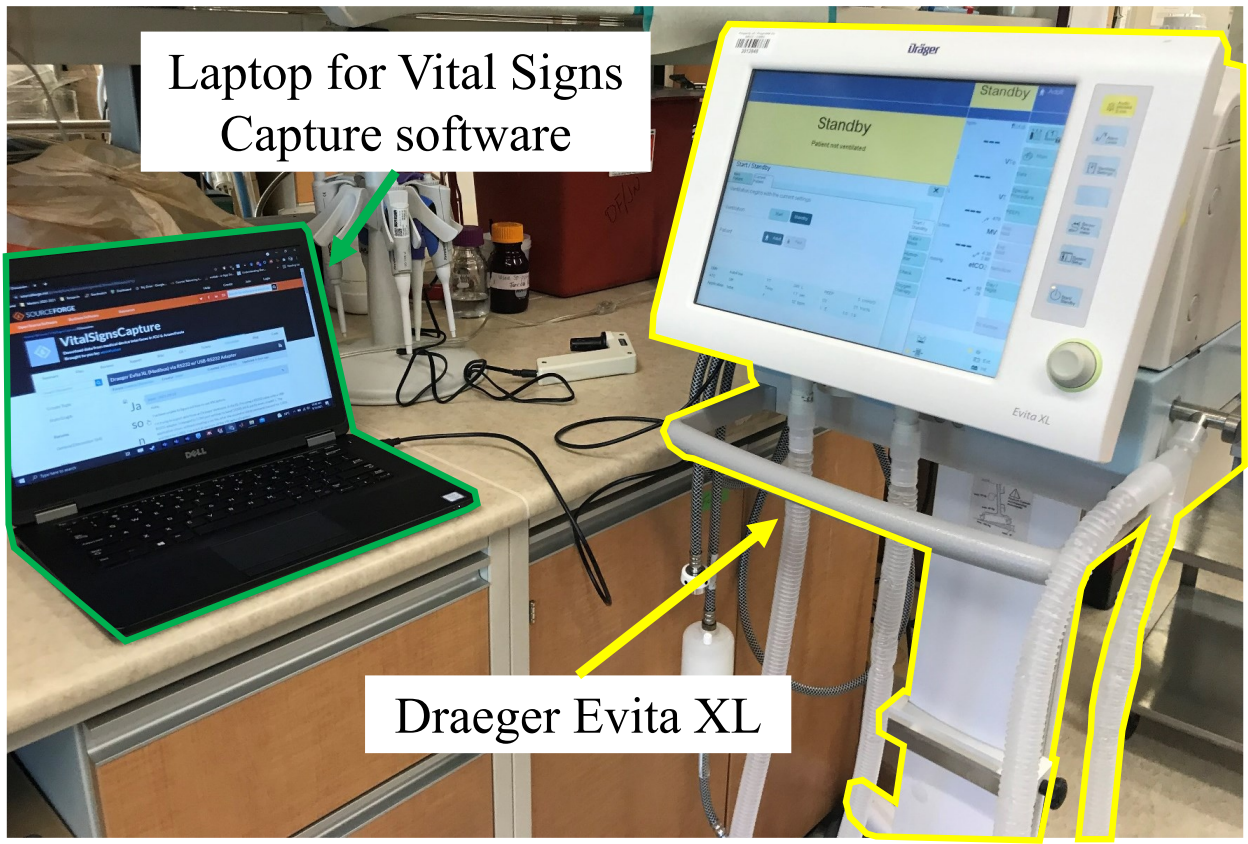


Figure 7.3 Annotated image of the Draeger Evita XL ventilator and laptop for Vital Signs Capture

The Draeger Evita XL, seen in Figure 7.3, was used to ventilate the porcine lung. It is a clinical ventilation unit for long term intensive care intended for adults, children, and neonates. The ventilator supports full mechanical ventilation or can assist spontaneous breathing. Ventilation can be performed in several different ventilation modes, including constant mandatory ventilation (CMV) which delivers breaths based on set variables, while regulating others. The mode, and breathing parameters are set using a touchscreen in a GUI. Additionally, it monitors metrics such as airway pressure, expiratory minute volume, and inspiratory tidal volume.

The Draeger Evita XL uses a proprietary encoding method, the Medibus protocol, for exporting data. Typically, specialized communication units are required to decode and display the ventilator's measurements and settings, however, the encoding method is publicly available. An open-source software called Vital Signs Capture, or VSCapture, was used to read measurements from the Draeger Evita XL using a StarTech RS232 to USB adapter to connect to a laptop as see in Figure 7.3.

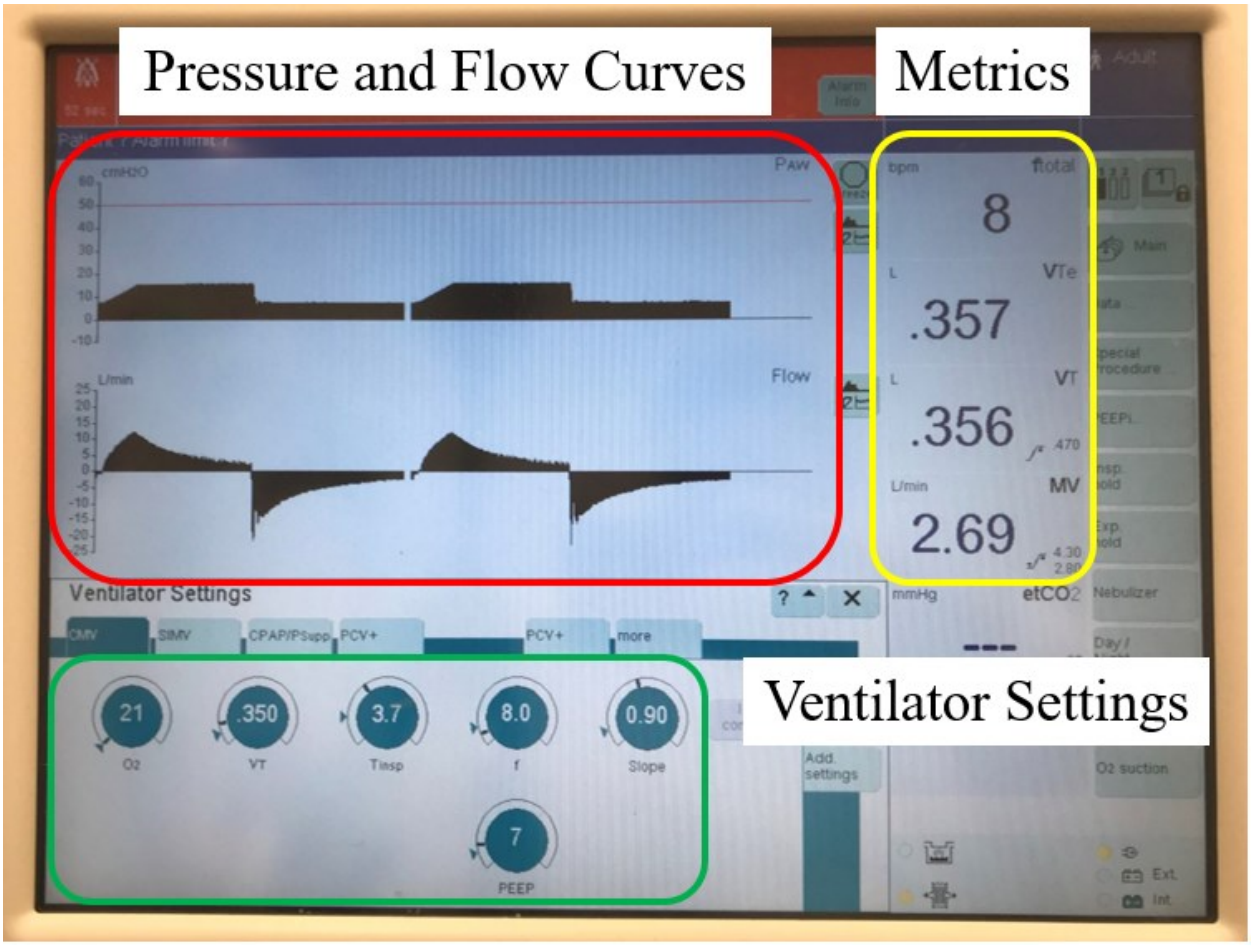


Figure 7.4 Annotated image of the Draeger Evita XL touchscreen GUI

The Draeger Evita XL uses a touchscreen and graphical user interface that is divided into three main sections, as shown in Figure 7.4. The first section at the bottom left of the screen allows the user to set ventilator settings, and the ventilation mode. The Draeger Evita XL is set in the CMV mode with six setting variables. The main ventilator settings shown are tidal volume (*VT*), inspiratory time (T_{insp}), breaths per minute (*f*), and slope, the time it takes to reach peak volume during inspiration. The second section, in the top right of the screen, provides real-time measurements including the expiratory tidal volume, average tidal volume, minute volume, and breathing rate. The last section, the top left of the screen, shows the airway pressure and flow rate curves. Other parameters are monitored, but not shown.

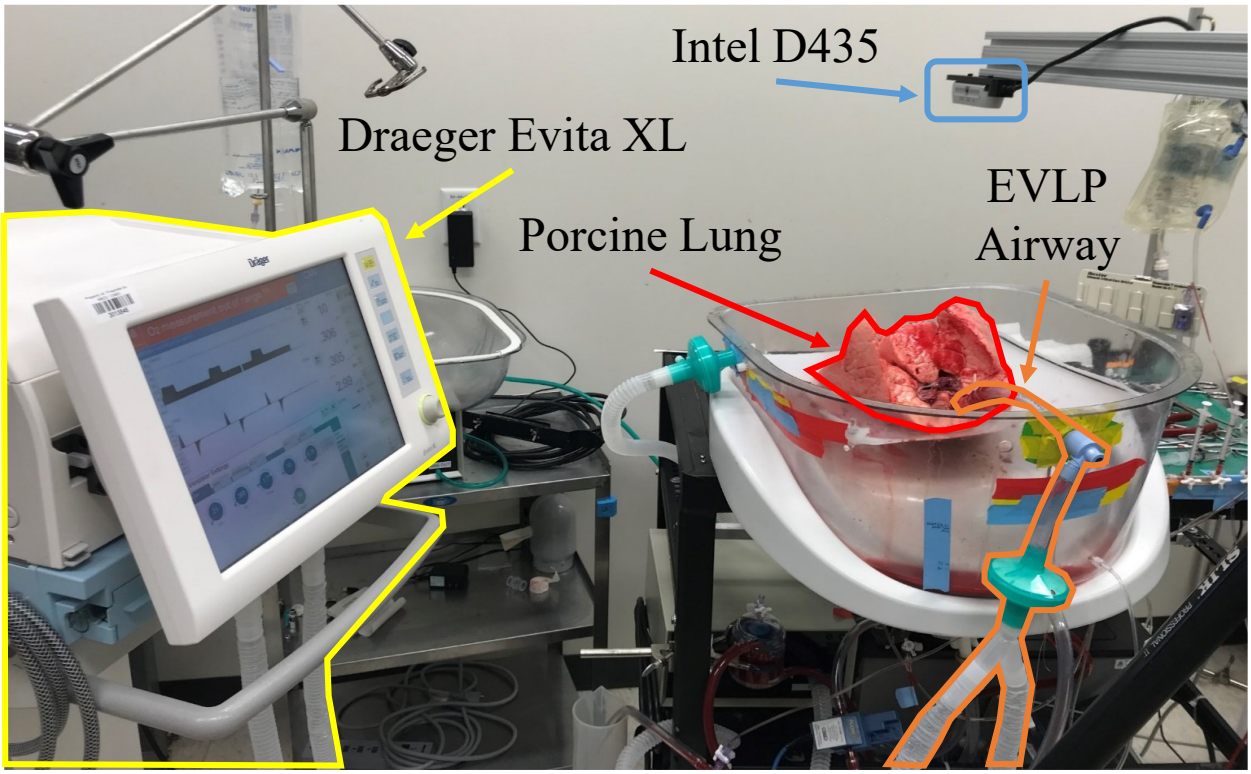
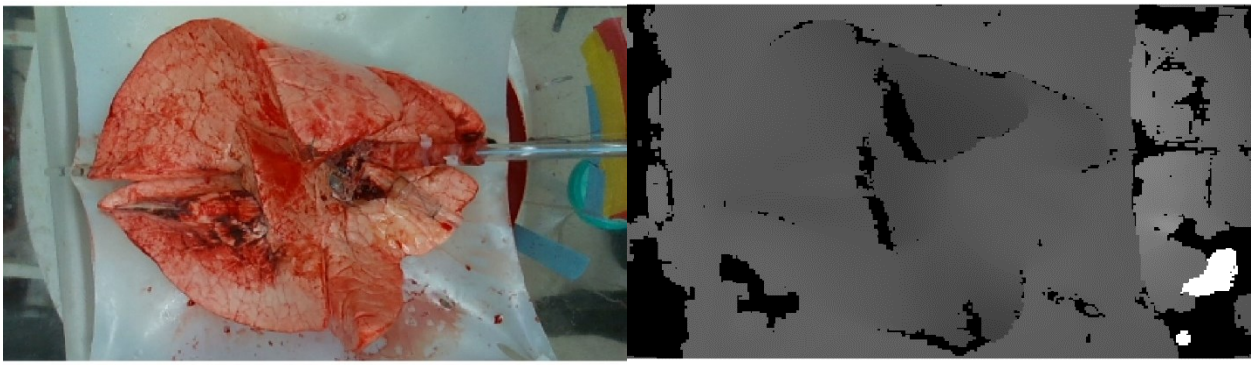


Figure 7.5 Annotated image of the porcine lung experimental setup

For this experiment, the Draeger Evita XL was connected to the porcine lung through the EVLP and intubated airway, as seen in Figure 7.5. It had several advantages for this experiment such as being mobile and having a publicly available manual. Also, most of its ventilation modes allow direct control over tidal volume, which the EVLP does not.

7.2 Calibration and Data Acquisition

Before data acquisition, the Intel RealSense D435 was calibrated using the same methods described in Chapter 4 and 6.2 . The intrinsic parameters, extrinsic parameters, and depth quality metrics from calibration are listed in Appendix D.



(a) *(b)*

Figure 7.6 Images of the porcine lung from the Intel RealSense D435 (a) color and (b) depth map streams

The color and depth map streams, as seen in Figure 7.6 (a) and (b), from the Intel RealSense D435 were recorded at 240 × 424 pixels and 30 fps, using the same methods described in Chapter 6.2 . Projective transformation was performed on the depth map to be aligned with the color image as seen in Figure 7.6. Additionally, the depth maps were spatially and temporally filtered using the Intel Realsense SDK 2.0. Default ASV settings were used in the Intel Realsense Viewer application. Notably, the image resolution was set to 240 × 424 because of the Intel Realsense Viewer would crash at higher resolutions likely due to memory limitations.

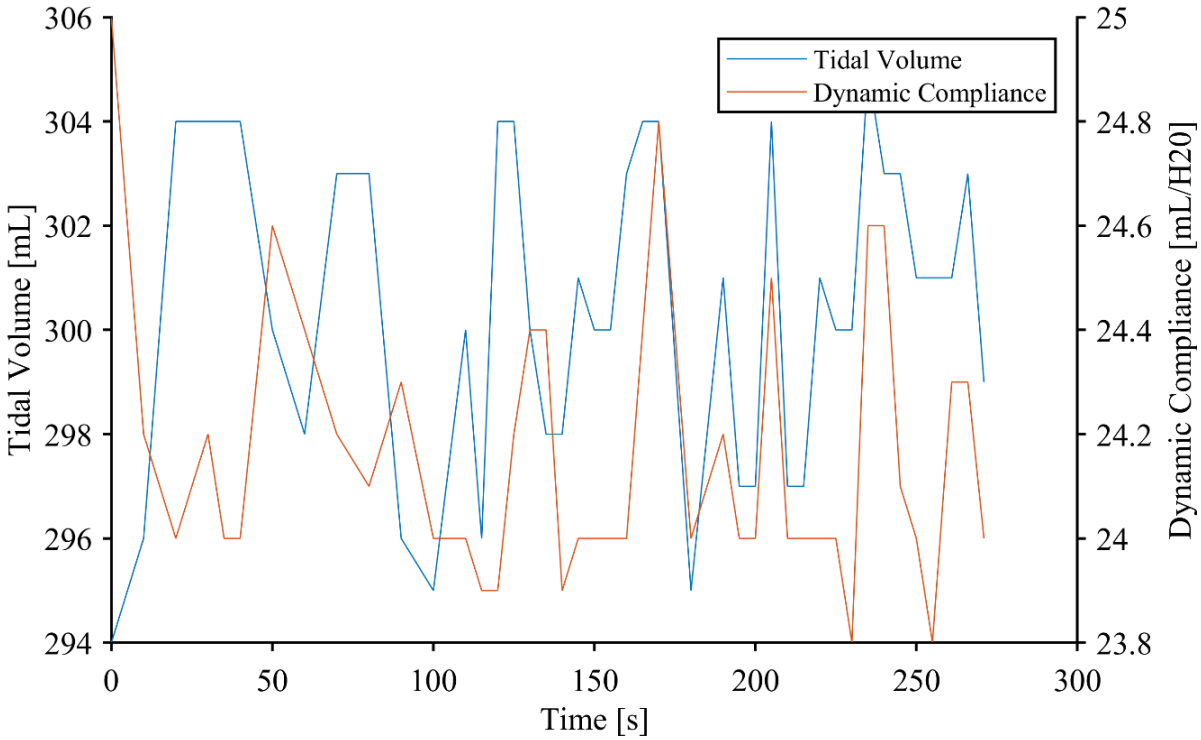


Figure 7.7 Plot of the tidal volume and dynamic compliance of the porcine lung during ventilation from the Draeger Evita XL obtained using Vital Signs Capture

The Draeger Evita XL measurements and set variables were recorded using a laptop running the Vital Signs Capture software in its real time mode, updating measurements every second. This includes tidal volume and dynamic compliance, as seen in Figure 7.7. The data was exported as three csv tables, listing information against timestamps, including ventilation mode, monitored metrics, and settings. Draeger Evita XL measurements did not include pressure or volume; however, *PIP* and *PEEP* were recorded.

Data was acquired from the Intel RealSense D435 and the Draeger Evita XL for several experiment cases with different ventilation settings. Experiment cases were designed to test the regional measurements and provide a larger range of values than the study in Chapter 6.

Table 7.1 Porcine lung experiment cases

Case	Experiment Type	Tidal Volume [mL/kg]	Tidal Volume [mL]
1	Steady state	4	300
2	Steady state	6	450
3	Steady state	8	600
4	Gradual Fall	8,7,6,5, and 4	600, 520, 450, 380, and 300
5	Gradual Rise	4,5,6,7, and 8	300, 380, 450, 520, and 600
6	Switch	4 to 8 to 4 …	300 to 600 to 300 …

Tidal volume was the only ventilation parameter to change between and within the six experiment cases, as seen Table 7.1. As mentioned in Chapter 6, it was assumed that only tidal volume would meaningfully change the measurements of the Intel RealSense D435, because it only measures the physical displacement of the porcine lung.

For cases 1, 2, and 3, the ventilation tidal volume remained constant to measure baseline performance. The tidal volume settings were selected based on the weight of the lung donor, which is common practice for mechanical ventilation [22], [142]. Typically, human lungs are ventilated with a low tidal volume between 6-8 mL per kg donor weight [142], with 12-15 mL per kg as a historic high [22]. These ventilation limits were used as a reference when the tidal volume levels were selected to avoid visual under inflation and over inflation of the porcine lung. Therefore, the tidal volume setting was found by multiplying the weight of the donor, 75 kg, by a tidal volume per weight.

The tidal volume changed within the other three experiment cases to measure changes in performance. They were designed to collect data to measure a wide range of tidal volume values, and measure changes with time. For Case 4 the tidal volume gradually decreased from 8 mL/kg to 4 mL/kg in 1 mL/kg increments, as seen in Table 7.1. Case 5 is the reverse of case 4, where the tidal volume was increased gradually from 4 mL/kg to 8 mL/kg. In case 6, the tidal volume was switched between 4 mL/kg and 8 mL/kg without incremental steps.

Since the Draeger Evita XL is controlled by setting a discrete target tidal volume, the tidal volume was manually changed for each increment in the experiment cases. However, the ventilator was found to take approximately 45 seconds to reaches a new target tidal volume after changing the setting. Therefore, data was recorded for about five minutes for each case, spending about 45 seconds at each tidal volume level when multiple levels existed.

Table 7.2 Porcine lung ventilator settings

Parameter	Setting
Inspiratory Time, [s]	2.0
f, [bpm]	8.0
Slope, [-]	0.90
PEEP, [cmH20]	10
O_2, [%]	21

These experiment cases were implemented by using the CMV auto-flow ventilation mode on the Draeger Evita XL. The CMV mode is entirely mechanical ventilation that strives to achieve set variables, while regulating others. The set variables include tidal volume, inspiratory time, breathing rate, slope, *PEEP*, and O_2 concentration by percentage. The set variables, except for tidal volume, remained the same between all experiment cases, as outlined in Table 7.2. Notably, the O_2 concentration setting was irrelevant because only pressurized air was connected to the Draeger Evita XL.

Typically, the ventilator is connected to pressurized medical grade oxygen and carbon dioxide using a gas mixer. The ventilator regulates this input airflow to achieve desired pressures or tidal volumes. However, for this experiment the ventilator was connected to pressurized air, and atmospheric air to simply pressurize the lungs, and achieve physiological displacement during breathing. Both medical grade oxygen and carbon dioxide were not needed or available, to achieve porcine lung displacement and gas exchange in tandem with blood perfusion.

7.3 Image Processing Scheme for Clinical Cases

7.3.1 Color Image and Depth Map Segmentation

In chapter 4, an edge-based method was used to segment the ventilator test lung, however, it was not re-used for the porcine lung. It was hypothesized that the edge-based method would fail to segment the porcine lung because its edges are not distinct in the depth map, as seen in Figure 7.6 (b). Also, the edge-based method in Chapter 6 was found to be dependent on depth holes caused by occlusion to separate the lung and the background, which the porcine lung depth maps lack. As an alternative, a region-based segmentation method was tested on the depth maps. However, this method was found to be inconsistent because the porcine lung and EVLP

trampoline are found at similar depths. Therefore, the other image streams from the Intel RealSense D435 were considered for segmentation.

The color image stream was the natural choice as an alternative image to segment the porcine lung. The edge and regions between the porcine lung and background are distinct based on color, as seen in Figure 7.6 (a). Therefore, both edge-based and region-based image segmentation methods were feasible. Also, projective transformation was performed on the depth maps to align them with the color images. Since they are aligned, the porcine lung will be in the same pixel locations in both images. Therefore, the segmentation binary map obtained from a color image can be used directly to segment the depth maps.

Through trial and error, the lazy snapping segmentation method was selected to segment the porcine lung in the color images. As described in Chapter 5, lazy snapping is a semi-automatic region-based method that segments an image into a foreground and background. It requires seed locations for the foreground and background that are interactively drawn shapes on top of the image.

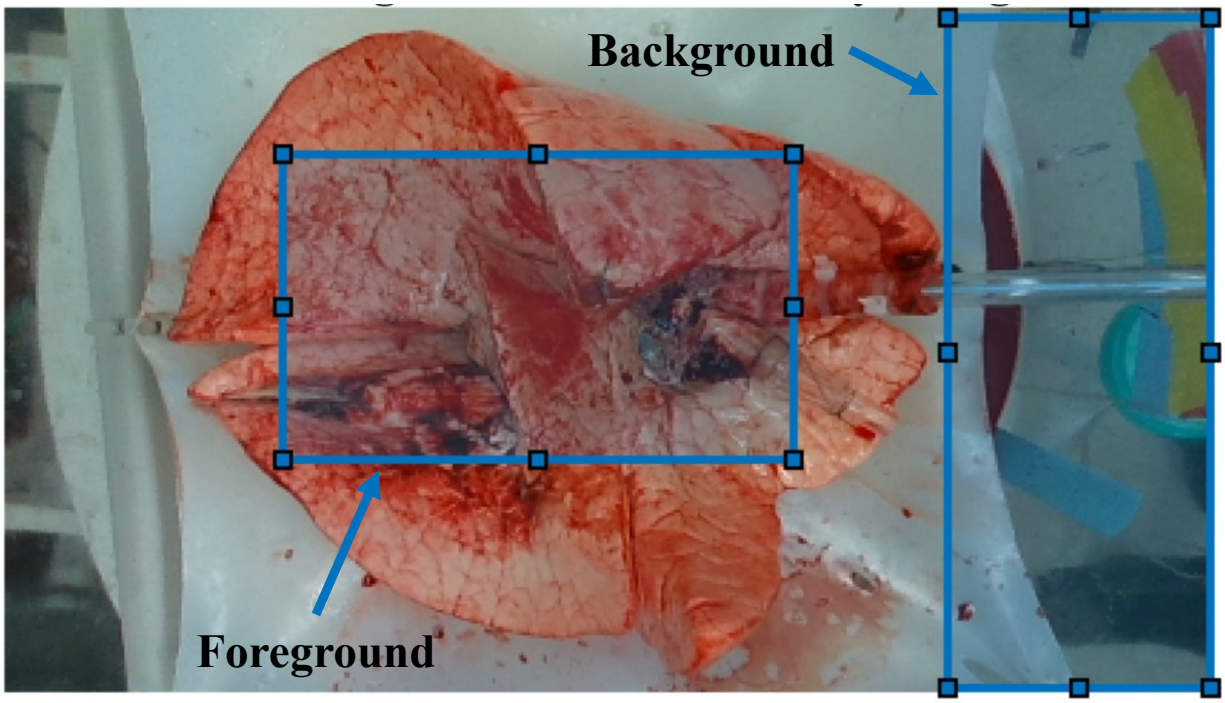


Figure 7.8 Annotated color image from the Intel RealSense D435 of the porcine lung with interactively marked foreground and background for segmentation

Foreground and background seed regions were interactively drawn on the color images, as seen in Figure 7.8. The foreground seed region is centered on and only includes the porcine lung. The background seed region selects the entire left side of the color image, not including the porcine lung. This section of the image includes the floor, EVLP trampoline and chamber, and the intubated airway connection. The foreground and background are only drawn once and reused to segment all color images in an experiment case.

To improve lazy snapping segmentation, the color images are contrast enhanced using histogram equalization [100], and the unsharp mask technique [100] to create more distinct edges. Also, the color images were oversegmented into sub-regions to improve computational performance [107].

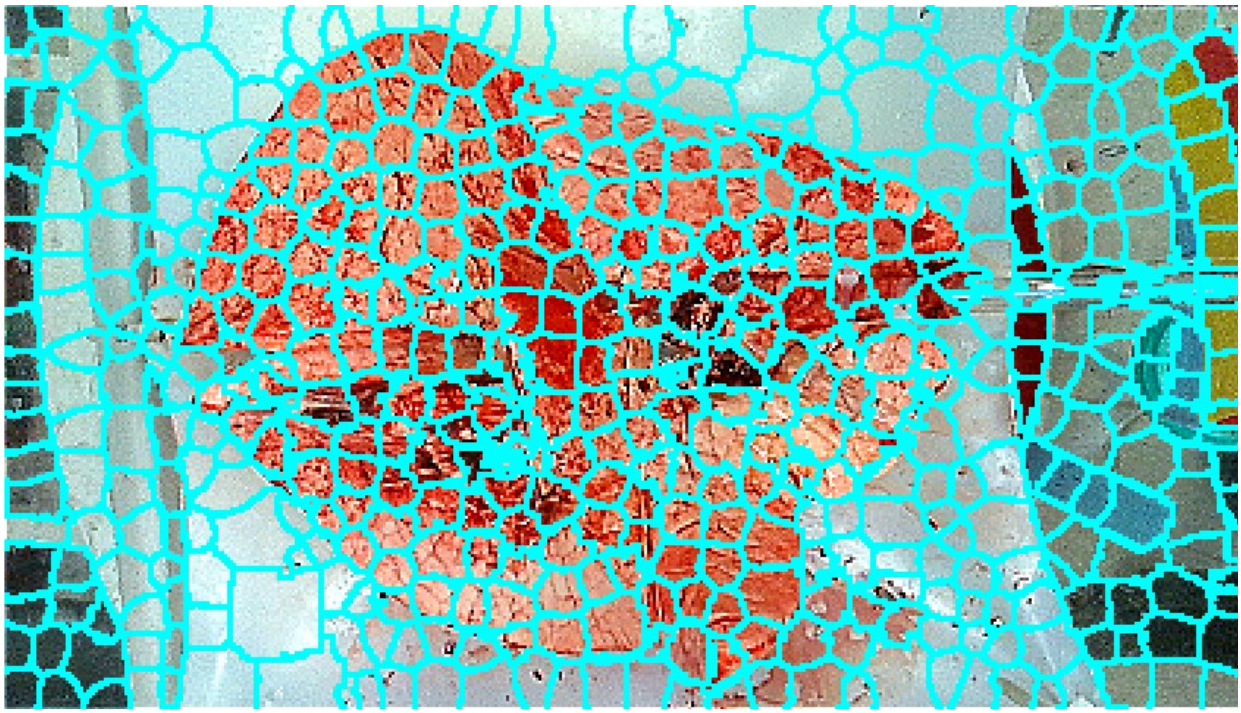
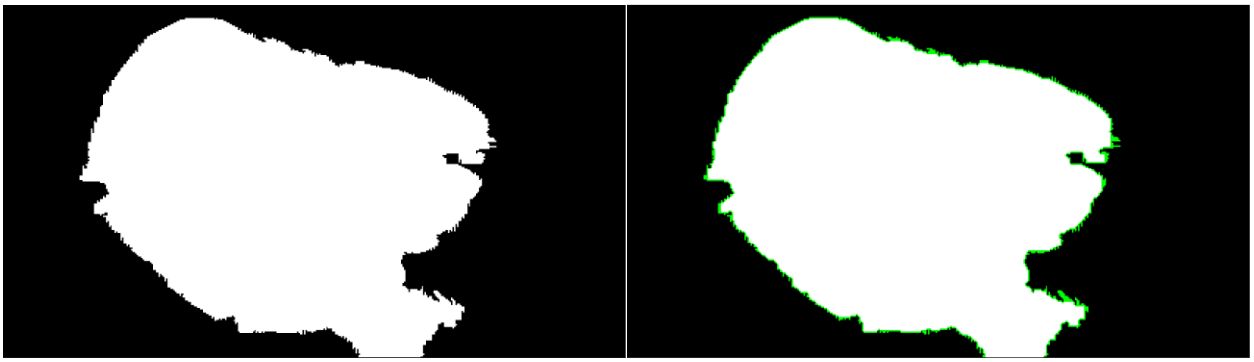


Figure 7.9 Color image of the porcine lung oversegmented into super pixels

Oversegmentation was performed on the color images using the super pixels method, which are semi-large groups of similar pixels [143], [144], as seen in Figure 7.9. Instead of lazy snapping each pixel, the method evaluates each super pixel as part of the foreground or background. Notably, the super pixels have edges that follow the perimeter of the lung, and other objects.



(a) *b)*

Figure 7.10 Image of a binary map segmented from the porcine lung color images (a) before and (b) after morphological closing

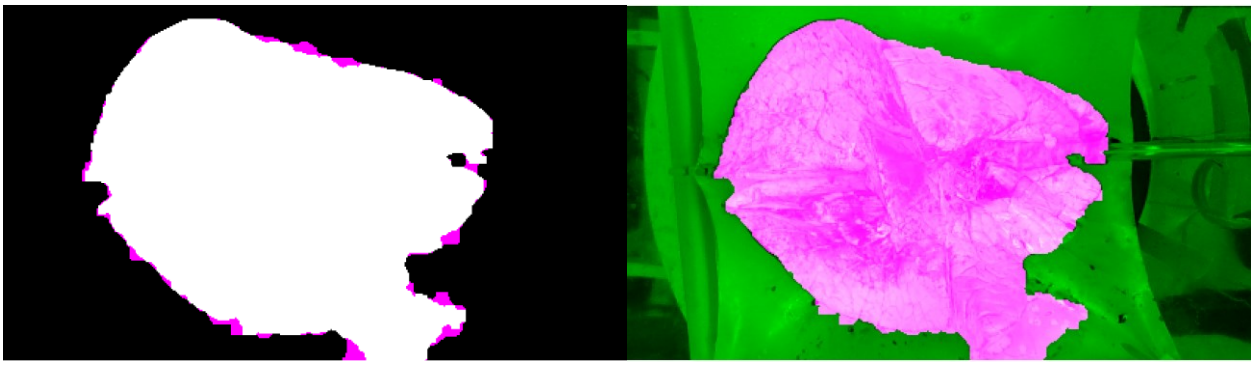
The porcine lung was segmented using lazy snapping with the super pixels on the color images, providing a binary map of the lung, as seen in Figure 7.10 (a). The binary lung retained the shape of the super pixels, which is visible along the lung's perimeter. However, the super pixels only

roughly match the outline of the porcine lung, creating spurs in the binary map. Also, the blood on the EVLP trampoline seen in Figure 7.8 was segmented along with the lung. To correct for these errors the binary maps were post-processed using four steps:

1. Object removal
2. Hole filling
3. Morphological smoothing
4. Temporal smoothing

In the first step, all bodies were removed from the binary maps, except the largest that was assumed to be the porcine lung. This step removed speckles of blood that were on the EVLP trampoline. However, this did not remove the blood right beside the lung. The second step was to fill any holes in the remaining body.

The third and fourth steps performed spatial and temporal smoothing, like the Intel RealSense SDK depth map filters. Spatial smoothing was performed using morphological closing with a circle structural element. This smoothed the perimeter of the porcine lung in the binary maps, removing the green pixels in Figure 7.10 (b).

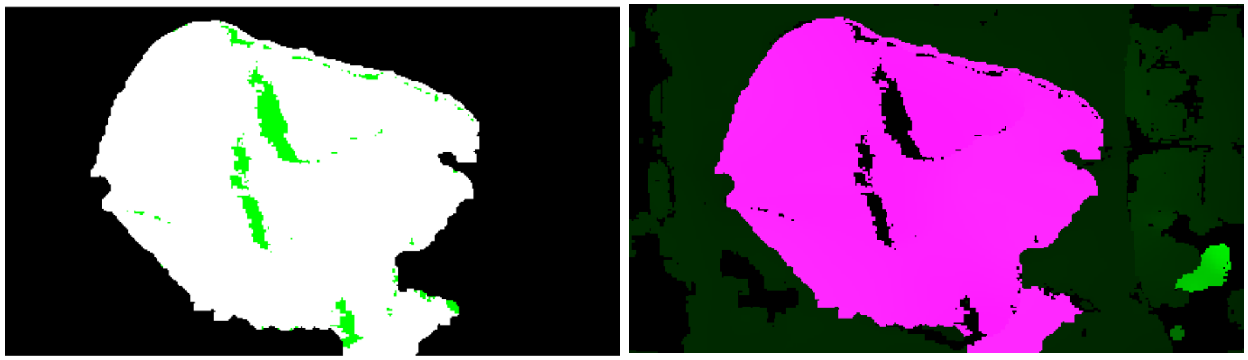


(a) *(b)*

Figure 7.11 Image of a binary map segmented from the porcine lung color images (a) after temporal filtering and (b) the original color image segmented by the binary map

Temporal smoothing was performed by finding the pixel-wise average across multiple binary maps, forming an image, then binarizing the result by thresholding. This method was used in Chapter 6.

The temporal smoothing process is repeated for all the binary maps with a moving window that selects images. If the sampling rate is known, the moving window size, moving average values, and threshold can be viewed in terms of seconds instead of frames. As a result, the binary maps only retain pixels that are consistently segmented, removing the magenta pixels in Figure 7.11 (a). These three parameters were adjusted through trial and error to obtain acceptable results. The temporally smoothed binary maps can be used to segment the porcine lung in the color images as seen in Figure 7.11 (b).



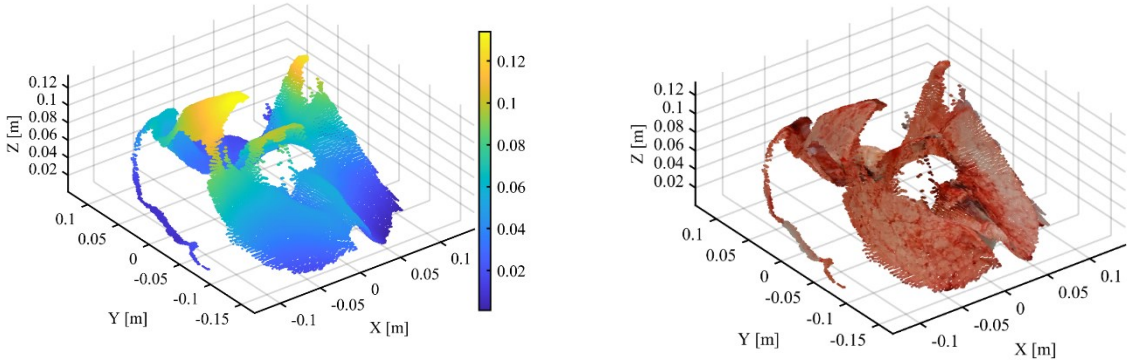
(a) *(b)*

Figure 7.12 Images of (a) the depth thresholded binary map of the porcine lung and (b) the depth map masked by the threshold binary map

To use the binary maps on the depth maps, pixels below a threshold in the depth maps are removed from binary maps. This step mostly removes pixels with depth holes, as seen in Figure 7.12 (a) and (b). The depth threshold was measured before the experiment, as the distance from the Intel RealSense D435 to the EVLP trampoline.

7.3.2 **Point Cloud Processing**

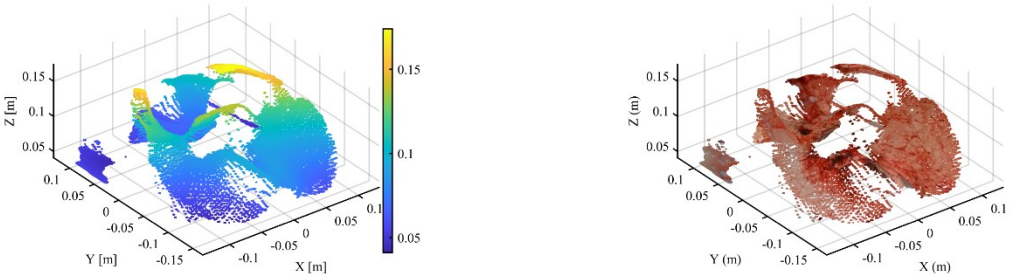
The same procedure described in Chapter 6.3.2 was performed on the porcine lung data.



(a) *(b)*

Figure 7.13 3D plot of the reconstructed surface of the porcine lung as (a) point cloud and (b) the point cloud colorized using the color images

The segmented depth maps were deprojected into point clouds, providing points on the surface of the porcine lung, as seen in Figure 7.13 (a). Also, the color image data was inherited by the point cloud, allowing the point cloud to be colored as seen in Figure 7.13 (b).



(a) *b)*

Figure 7.14 3D plots of the downsampled and transformed porcine lung (a) point cloud and (b) colored point cloud

The point clouds were downsampling using a box averaging filter to make it sparser, remove outliers, and denoise the surface. Also, the point clouds were transformations to align the point clouds with the EVLP trampoline, as see in Figure 7.14 (a) and Figure 7.14 (b).

7.3.3 **Surface Reconstruction of the Porcine Lung**

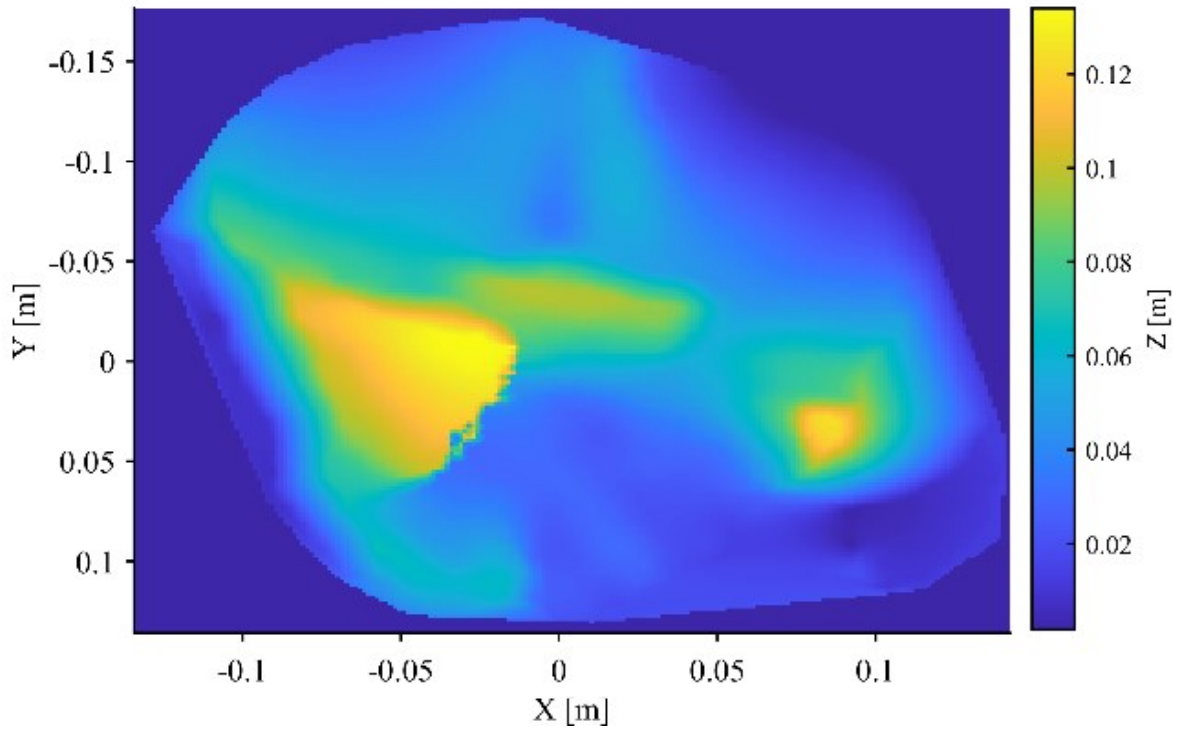


Figure 7.15 Plot of the surface map of the porcine lung

The porcine lung's surface was reconstructed from its point clouds, as described in Chapter 5. The spatial and color information was interpolated at the same grid points on the *xy* plane, for all frames from the down sampled point clouds, as seen in Figure 7.15.

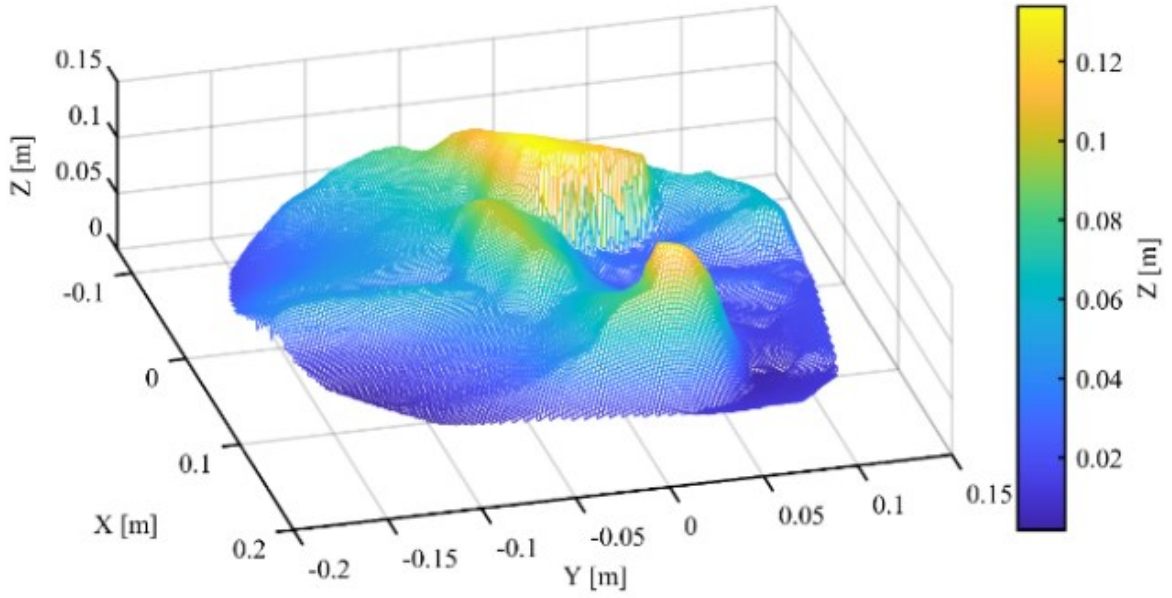
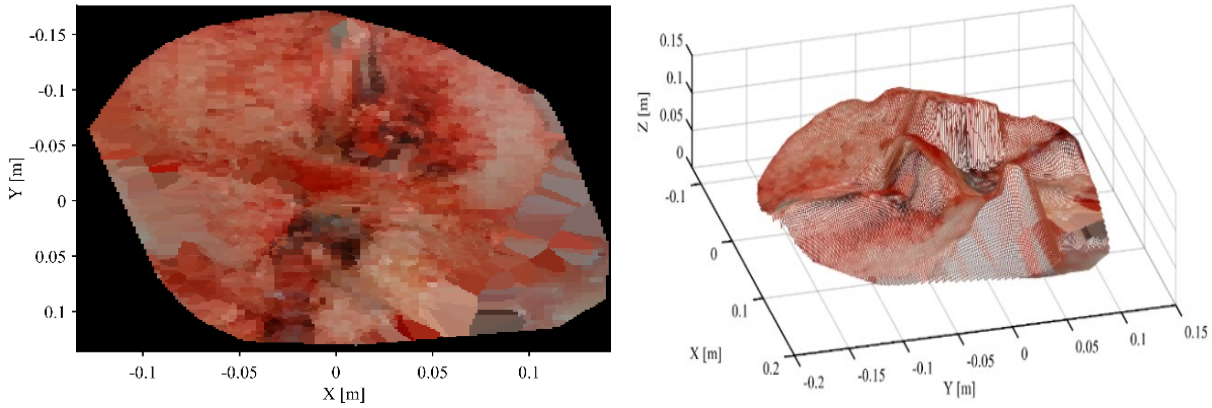


Figure 7.16 3D Plot of the Delaunay triangulation meshed surface map of the porcine lung

The interpolated points were meshed using Delaunay triangulation. Since the query grid points' *x* and *y* coordinates are constant between frames, the mesh is constant between frames, as seen in Figure 7.16.



(a) *(b)*

Figure 7.17 Plots of (a) the colored surface map of the porcine lung as a 2D plot and (b) 3D plot

The surface map can be colored using the interpolated color information, as seen in Figure 7.17 (a) and (b). The surface maps can be treated like images, allowing arithmetic operations within and between frames, thus digital image processing. Also, the surface maps can be used to measure whole lung plethysmography measurements such as displacement using the Divergence Theorem.

7.3.4 **Plethysmography Measurements of a Porcine Lung**

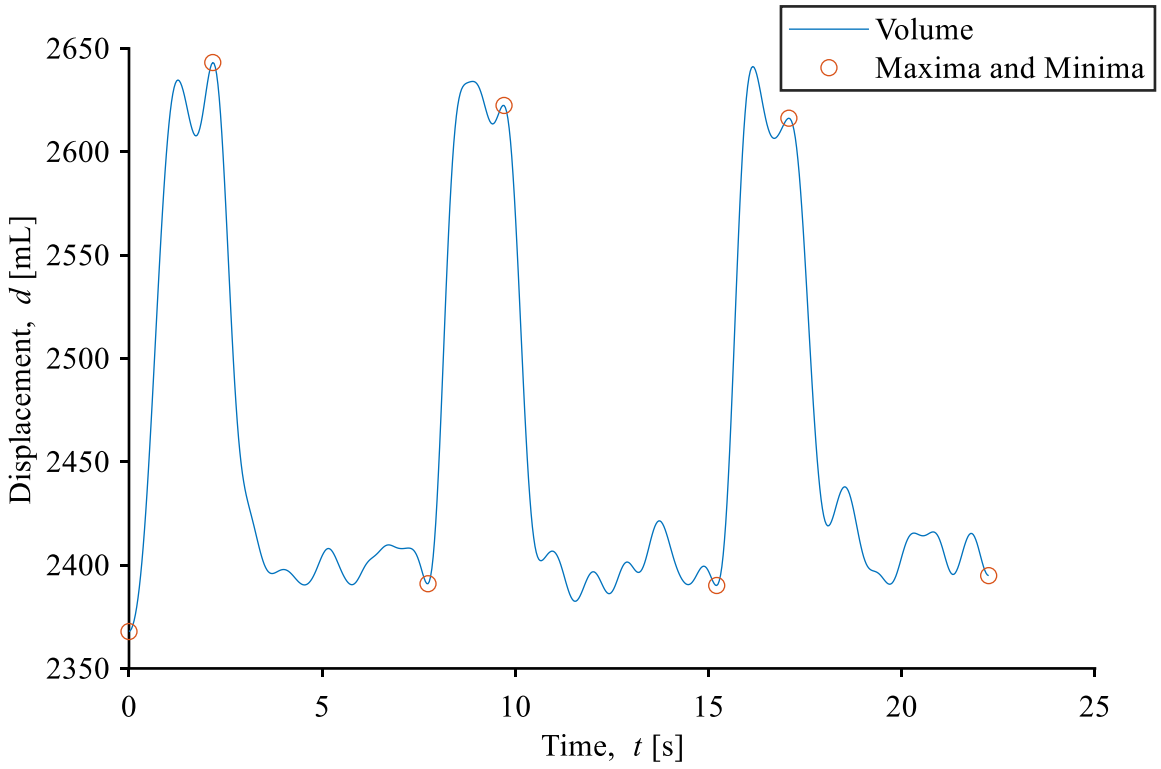


Figure 7.18 Plot of the ASV displacement of the porcine lung sampled from case 1

The same processing method described Chapter 5 and implemented in Chapter 6 were used to derive plethysmography measurements from the interpolated surfaces of the porcine lung. The volume, or displacement, of the porcine lung was estimated using the Divergence Theorem described in Chapter 5. From this signal, the inhale and exhale local extrema points can be segmented using the second derivative zero-crossing method. Figure 7.18 is the estimated volume of the porcine lung from the interpolated surfaces, with the segmented inhale-exhale local extrema points. This allows for the calculation of metrics such as tidal volume, and inspiratory time.

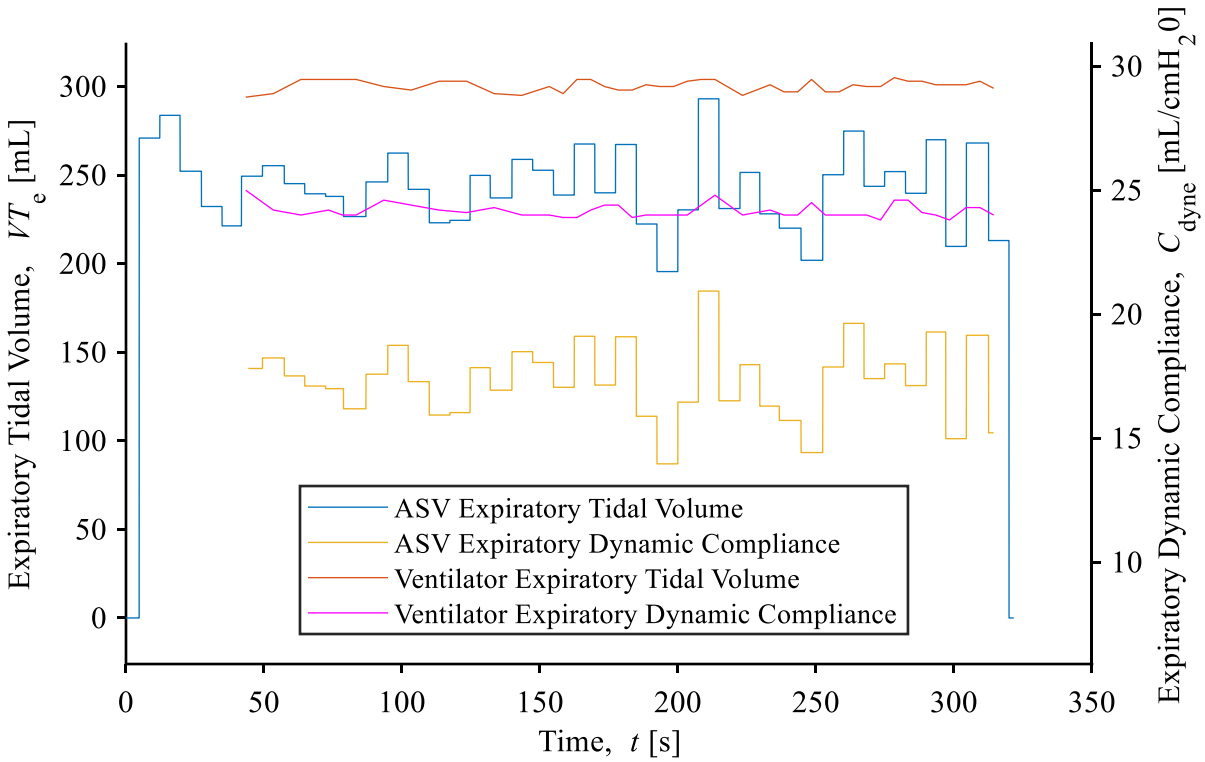


Figure 7.19 Plot of the ASV and ventilator porcine lung tidal volume and dynamic compliance of case 1

The Draeger Evita XL *PIP* and *PEEP* measurements were used to calculate dynamic compliance from the tidal volume measurements. To combine the ventilator and Intel RealSense D435 measurements, the ventilator measurements were interpolated to match the timestamps of the Intel RealSense D435 data, as seen in Figure 7.19. Notably, the *PIP* and *PEEP* were constant for the steady state experiment cases, explaining the similarity between the calculated tidal volume and dynamic compliance. The results showed that the ASV method had significantly more variance than the ventilator, as seen in Figure 7.19.

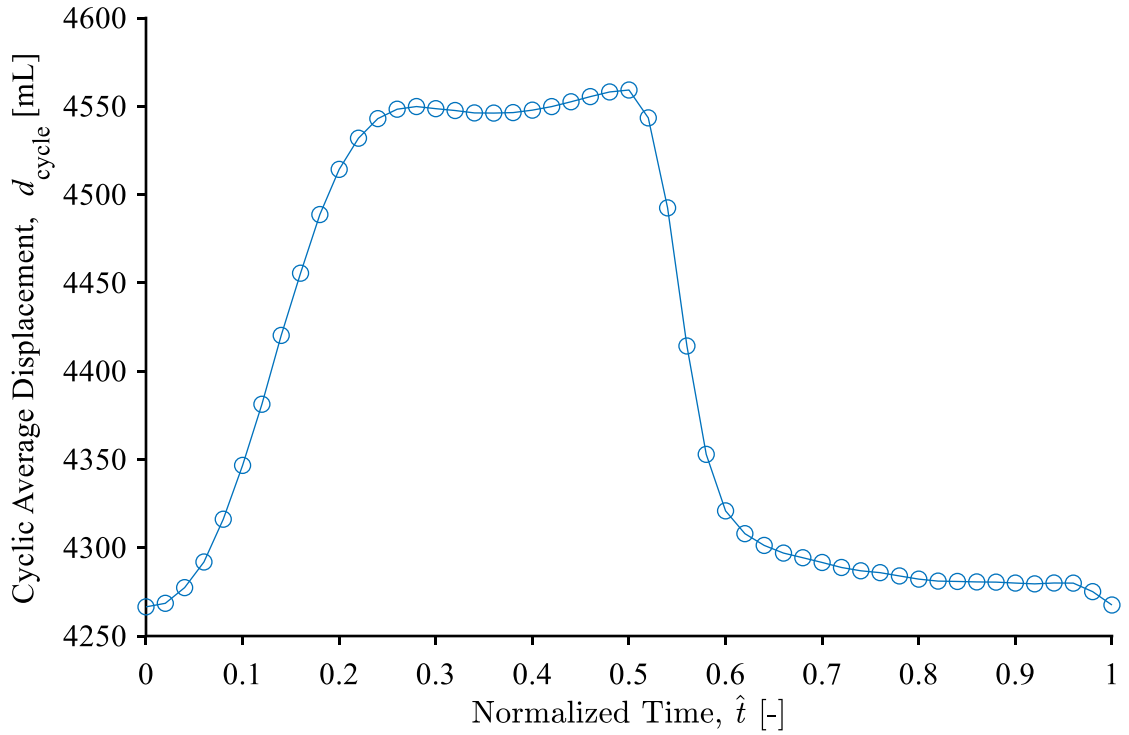


Figure 7.20 Plot of the cyclic average displacement of the porcine lung of case 1

The baseline performance of the porcine lung was measured as a cyclic average of the displacement signal, as seen in Figure 7.20. This processing scheme averages all the segmented breaths, into an averaged volume relative to a normalized time within a cycle. These results characterized the porcine lung displacement to measure deviations in performance.

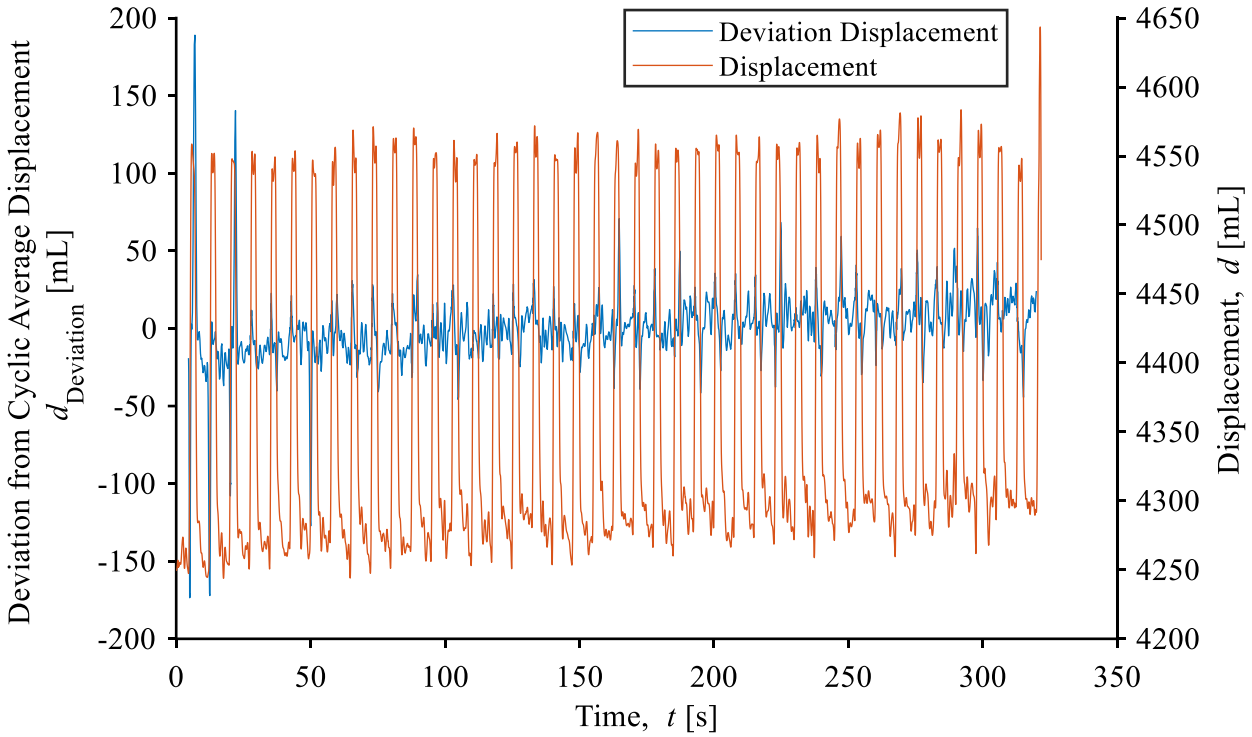


Figure 7.21 Plot of the displacement and cyclic average deviation of the porcine lung from case 1

The change in performance of the porcine lung can be tracked by measuring the deviation from the cyclic average displacement. This deviation was found using *compare_to_avg_cycle()* in Appendix B. As seen in Figure 7.21, this deviation provides feedback like a trend line, but allows the user to understand if the lung is under or over performing relative to its average displacement. In general, this trend line showed the same ventilation patterns for each case as described in Table 7.1.

7.3.5 Regional Measurements of a Porcine Lung

The interpolated surfaces, in Chapter 7.3.3, were used to measure shape change of the porcine lung. They were used to create surfaces that effectively measure standard plethysmography measurements like tidal volume and dynamic compliance.

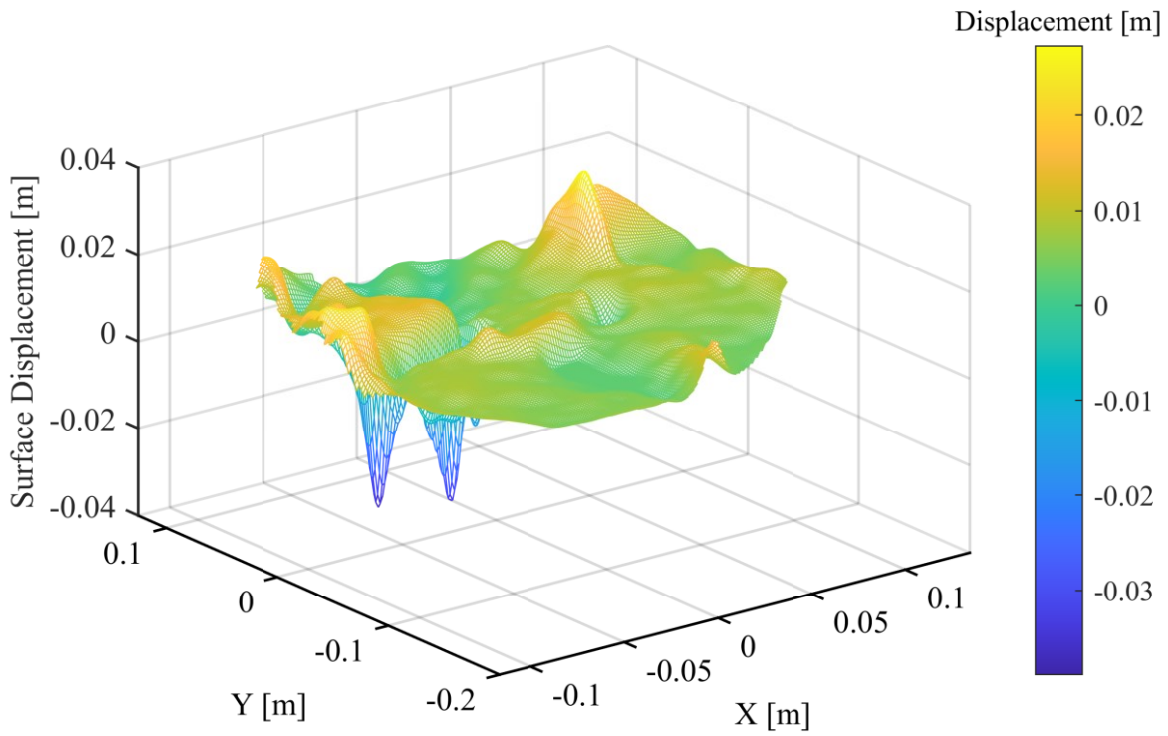


Figure 7.22 3D plot of surface tidal displacement of the porcine lung from case 1

The surface tidal displacement of the porcine lung was found as the difference between surface maps, as seen in Figure 7.22. For most cases, the porcine lung was found to have a consistent surface tidal displacement shape, as the magnitude in displacement increased with ventilation tidal volume. These surfaces indicated that the porcine lung had generally uniform regional displacement except for the peaks of the left and right lung lobes in yellow, and the local minima in blue that are likely artifacts caused by horizontal motion of the entire lung.

The surface dynamic compliance was found by scaling the surface tidal displacement by the difference of the *PIP* and *PEEP*. The performance and health of the lung for a region, can be measured by the surface dynamic compliance.

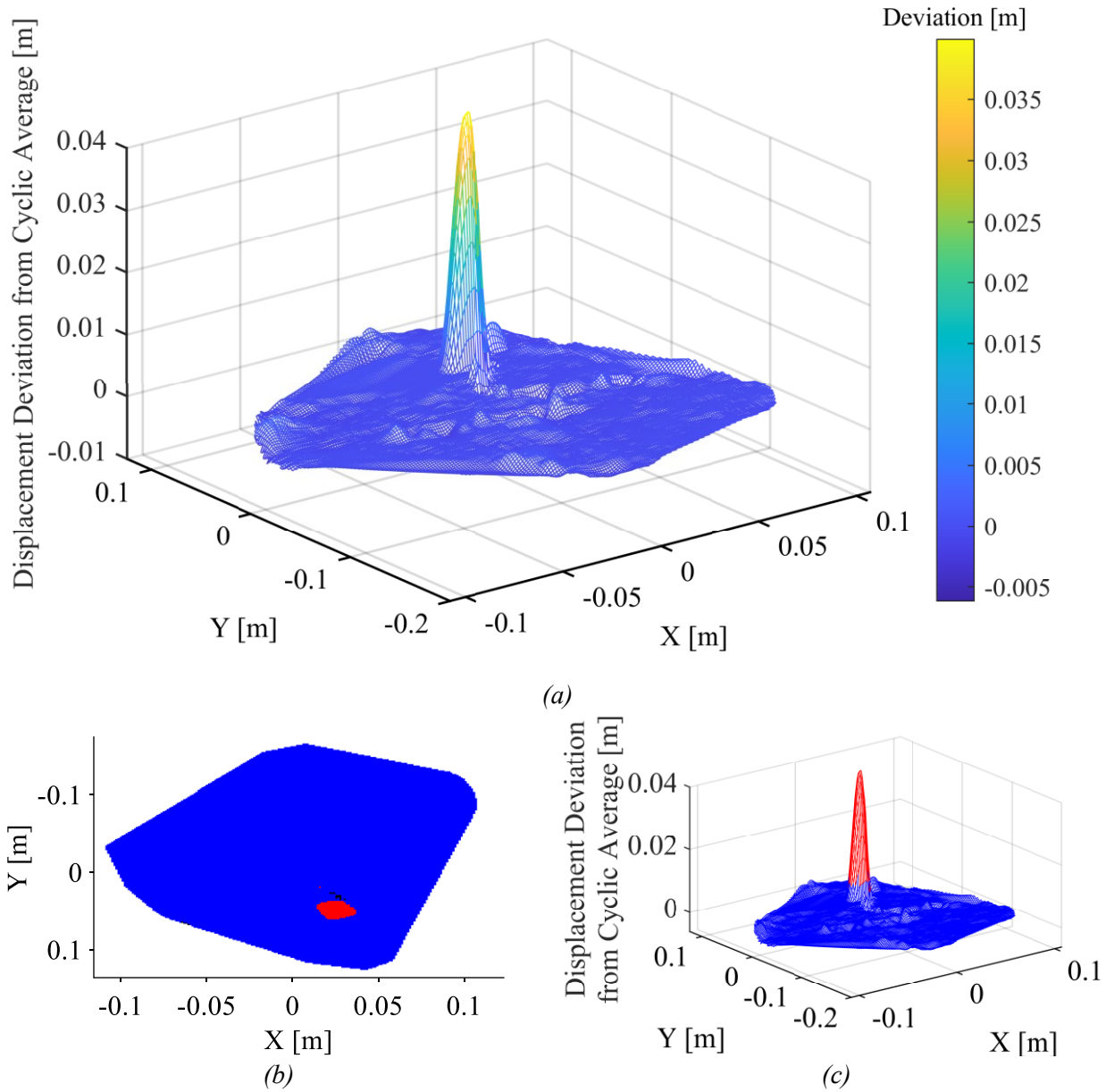


Figure 7.23 Plots of (a) the surface deviation from cyclic average displacement of the porcine lung from case 1 for peak detection (a) as an image and (b) as a surface mesh

The surface deviation, seen in Figure 7.23 (a), from the surface cycle average displacement of the porcine lung was found using the method described in Chapter 5. The surface tidal displacement and deviation were checked for local minima and maxima to identify potential locations of over and under inflation. In case 1, a local maxima was found in the surface deviation marked as red in Figure 7.23 (b) and (c), where blue indicates not a minima or maxima. These results indicate that the porcine lung may have localized over-inflation.

7.4 Comparison of Measurements

The same method comparison analysis in Chapter 6.4 was used with the porcine lung measurements. The ASV method was evaluated based on measurement distribution, correlation, and agreement.

Unlike Chapter 6, data was collected from transient state cases, in which the tidal volume setting changed periodically. These transient cases provided a range of measurements that were valid for correlation and agreement analysis, which was a limitation in Chapter 6.4 . Also, the study was limited to only comparing tidal volume and dynamic compliance measurements.

7.4.1 Preparation of Measurements for Comparison

The inspiratory tidal volume and dynamic compliance were the only measurements used to compare the two methods. As previously mentioned, the VSCapture software did not record any other mutual measurements. Also, it did not record pressure, volume, or flow rate that could have been used to derive other plethysmography metrics.

The six experiment cases were analyzed individually, and as two groups. The steady state group was comprised of cases 1, 2, and 3, while the transient state group was cases 4, 5, and 6. Some cases or groupings were omitted depending on the analysis step. Invalid measurements, such as NAN and INF values in MATLAB, were removed before any analysis. Also, VSCapture recorded less measurements, one per breath, than the Intel RealSense Viewer, despite VSCapture and Intel Realsense Viewer starting data acquisition at the same time. This means some measurements from the ventilator were missing, so there are unpaired ASV measurements. To correct for this error, the unpaired measurements were removed from all analysis.

7.4.2 Measurement Distribution of the ASV and Ventilator Systems

The measurement behavior of the ASV and ventilator systems were evaluated and compared for each experiment case. Specifically, their measurement means were compared to the tidal volume setting, and each other. Also, the ASV method's repeatability was evaluated by measuring its standard deviation. Kurtosis and skewness, accompanied by tailedness and symmetry, of the two systems were found to identify any tendencies. These findings were compared to observations

made from their histograms and probability plots. Lastly, the data sets were checked for normality using the Shapiro-Wilk tests.

Table 7.3 Measurement distribution mean error of the ASV and ventilator systems for the porcine lung

Experiment Case	Inspiratory Tidal Volume Setting, *VTi* [mL]	*VTi* Mean [mL]		Error Relative to *VTi* Setting [%]		
		ASV	Ventilator	ASV	Ventilator	ASV - Ventilator
1	300	291.99	300.21	-2.67	0.07	-2.74
2	450	607.57	480.96	35.02	6.88	26.38
3	600	920.54	627.70	53.42	4.62	48.81
4	450	772.57	505.35	71.68	12.30	59.38
5	450	665.20	443.16	47.82	-1.52	49.34
6	450	725.53	471.80	61.23	4.84	56.38

To evaluate the accuracy of the VSCapture measurements, its means were compared to ventilation tidal volume set for each steady state case. The relative error for cases 1, 2, and 3 suggest that the VSCapture measurements are overestimating, or the ventilation is overshooting by 5%, as seen in Table 7.3. This suggests the ventilator measurements are close to the true values and can be used to evaluate the ASV method. In comparison, the ASV's error suggests it has error up to 50% and proportional bias, since error increases with ventilation tidal volume. The mean error analysis was performed on the transient cases as well.

Since the ventilation tidal volume changes within each transient case, the median of the ventilation setting range, 450 mL, was approximated as its mean. This assumption should be valid if the porcine lung was ventilated at each tidal volume setting for the same duration. The error for the transient cases showed no pattern, except that the ventilator has much lower error than the ASV, as seen in Table 7.3.

Table 7.4 Mean and standard deviation of the porcine lung tidal volume and dynamic compliance measurements from the ASV and ventilator for all experiment cases

Parameter	Experiment Case	Mean			Standard Deviation		
		ASV	Ventilator	Error [%]	ASV	Ventilator	Error [%]
Inspiratory Tidal Volume, *VTi* [mL]	1	291.99	300.21	-2.74	14.46	2.69	438.10
	2	607.57	480.96	26.32	55.58	2.28	2335.03
	3	920.54	627.70	46.65	65.91	3.64	1710.56
	4	772.57	505.35	52.88	199.77	115.28	73.30
	5	665.20	443.16	50.11	204.01	116.52	75.08
	6	725.53	471.80	53.78	232.00	133.69	73.53
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	1	20.86	24.18	-13.73	1.03	0.22	378.17
	2	24.34	20.77	17.18	2.22	0.23	859.54
	3	23.58	17.68	33.35	1.70	0.14	1143.44
	4	31.27	22.67	37.92	5.97	4.25	40.34
	5	32.04	24.34	31.64	7.05	4.30	64.06
	6	30.94	23.26	33.04	7.04	5.00	40.64

The mean and standard deviation error was measured for tidal volume and dynamic compliance between the ASV and ventilator measurements for all individual cases, as seen in Table 7.4. The steady and transient state case mean error results suggest the ASV system has a large mean offset, between 30% and 50% error. Also, the steady state case error increases with ventilation tidal volume, indicating proportional bias for both metrics. These results match the results in Table 7.3.

The standard deviation errors suggest the ASV has poor repeatability relative to the ventilator, since it is up to 20 times greater, as seen in Table 7.4. Also, the ASV system was observed to have proportional standard deviation, unlike the ventilator system.

Overall, the errors suggest the ASV has significant error and proportional bias. Also, the tidal volume mean errors were greater than the dynamic compliance, but this is the reverse for standard deviation errors. Lastly, case 1 always had the lowest error for each metric and parameter combination.

Table 7.5 Shapiro Wilk hypothesis test for normality and p-value for the measurement distribution of inspiratory tidal volume and dynamic compliance of the porcine lung

Experiment Case	Inspiratory Tidal Volume, *VTi*		Inspiratory Dynamic Compliance, C_{dyni}	
	ASV	Ventilator	ASV	Ventilator
1	✓	×	✓	×
2	×	×	×	✓
3	✓	✓	×	✓
4	×	×	×	×
5	×	×	×	×
6	×	×	×	✓

To identify any measurement tendencies, Shapiro-Wilk hypothesis tests checked the measurement distributions for normality, as seen in Table 7.5. Most experiment cases were non-normal with a few exceptions that showed no patterns.

Table 7.6 Kurtosis and skewness of the porcine lung tidal volume and dynamic compliance measurements from the ASV and ventilator for all experiment cases

Parameter	Experiment Case	Excess Kurtosis			Skewness		
		ASV	Ventilator	Error [%]	ASV	Ventilator	Error [%]
Inspiratory Tidal Volume, *VTi* [mL]	1	-0.43	-0.82	-47.15	-0.06	-0.35	-83.37
	2	2.99	2.23	34.24	1.77	-0.75	-337.42
	3	2.70	-0.48	-664.99	-0.72	0.04	-1728.33
	4	-1.41	-1.26	11.69	-0.20	-0.34	-40.93
	5	-1.04	-1.34	-22.33	0.19	0.28	-31.71
	6	-1.60	-1.71	-6.60	0.07	-0.01	-765.14
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	1	-0.43	0.87	-149.93	-0.06	1.01	-105.80
	2	2.95	-0.49	-703.43	1.75	-0.09	-2061.14
	3	2.50	0.02	10354.64	-0.68	-0.42	62.02
	4	0.81	-0.92	-187.29	1.21	0.55	120.80
	5	0.62	-1.42	-143.41	0.88	-0.03	-3194.69
	6	-0.18	-1.71	-89.60	0.40	0.19	102.93

Kurtosis and skewness were measured for tailedness and symmetry to compare with any trends observed in the histograms and probability plots. Excess kurtosis and skewness measurements, as seen in Table 7.6, which shows the ASV would tend to have more outliers and is more left-tailed than the ventilator since it has higher values for both metrics. Also, these results suggest that the ASV deviates more from normality than the ventilator.

Table 7.7 Tailedness and symmetry of the porcine lung tidal volume and dynamic compliance measurements from the ASV and ventilator for all experiment cases

Parameter	Experiment Case	Tailedness		Symmetry	
		ASV	Ventilator	ASV	Ventilator
Inspiratory Tidal Volume, *VTi* [mL]	1	Mesokurtic	Mesokurtic	Symmetric	Symmetric
	2	Leptokurtic	Leptokurtic	Left-Tailed	Symmetric
	3	Leptokurtic	Mesokurtic	Symmetric	Symmetric
	4	Platykurtic	Platykurtic	Symmetric	Symmetric
	5	Platykurtic	Platykurtic	Symmetric	Symmetric
	6	Platykurtic	Platykurtic	Symmetric	Symmetric
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	1	Mesokurtic	Mesokurtic	Symmetric	Left-Tailed
	2	Leptokurtic	Mesokurtic	Left-Tailed	Symmetric
	3	Leptokurtic	Mesokurtic	Symmetric	Symmetric
	4	Mesokurtic	Platykurtic	Left-Tailed	Symmetric
	5	Mesokurtic	Platykurtic	Symmetric	Symmetric
	6	Mesokurtic	Platykurtic	Symmetric	Symmetric

When each case is classified by excess kurtosis and skewness, as seen in Table 7.7, it shows that the ASV and ventilator have similar tailedness and symmetry. One of the few exceptions is the ASV's dynamic compliance tailedness for the transient cases, which is oddly mesokurtic or has a similar tailedness to a normal distribution.

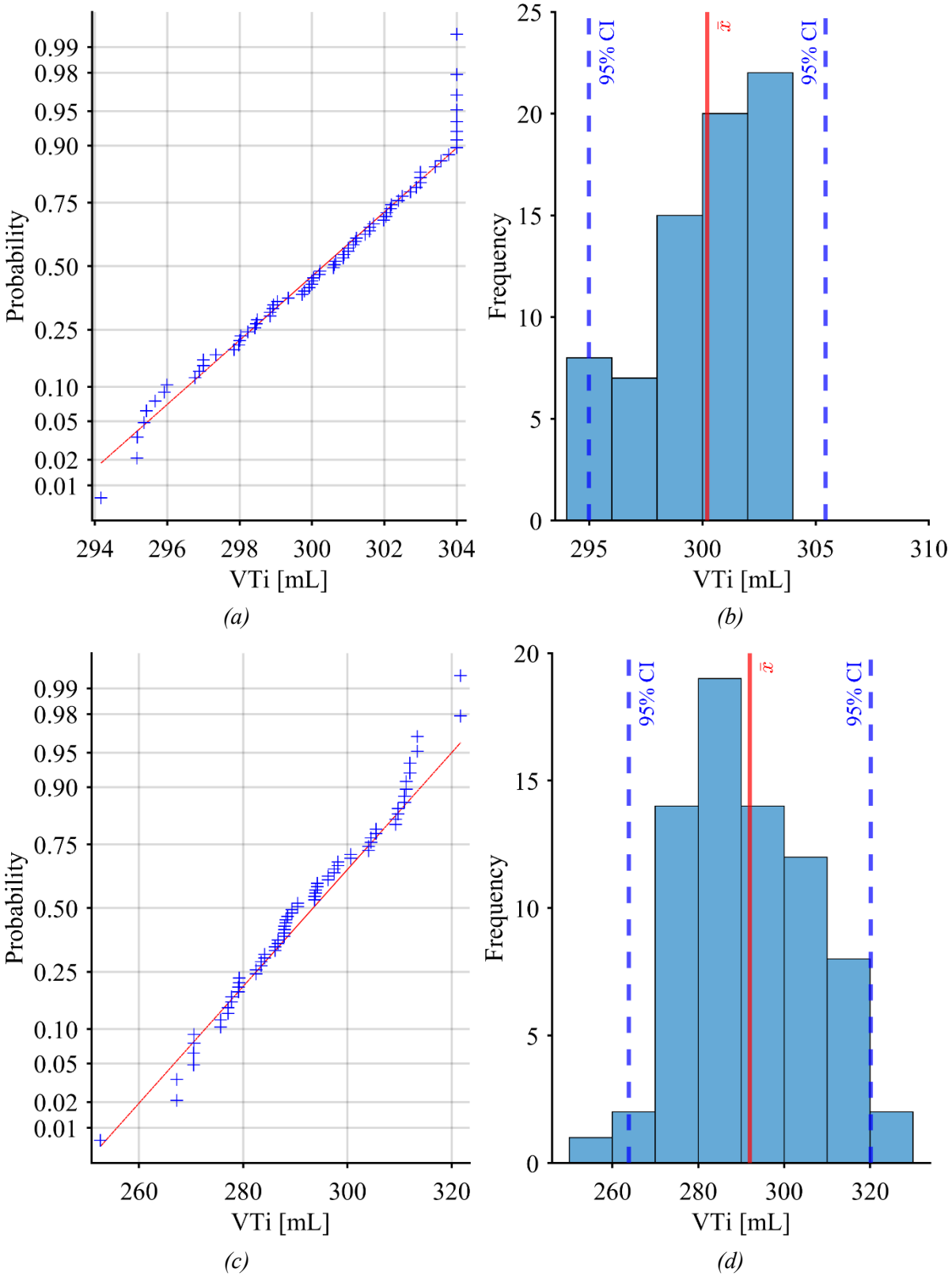


Figure 7.24 Probability plots and histograms of the inspiratory tidal volume measurement distribution from the ventilator(a) and b) and ASV (c) and (d) system from experiment case 1

The probability and histogram plots suggest the ASV and ventilator have three main differences, as seen in Figure 7.24. Firstly, the systems had different measurement distribution shapes, for the same parameter and experiment case. Secondly, the ASV measurements deviated from normality more than the ventilator based on the probability plots, but most did not appear to be normally distributed. Thirdly, the two systems had significant mean offset and different ranges for the same parameter. Of all these observations, most concur with the previous distribution analysis steps.

In this sub-section the ASV and ventilator measurement distributions were evaluated and compared in several different analysis. First, the distribution means and standard deviations were measured, and when compared suggests that the ASV has a large amount of dispersion, mean offset, and proportional bias. Next, normality was tested for using Shapiro-Wilk hypothesis tests that found most were not normal with no pattern. Excess kurtosis and skewness were measured, which found the ASV system to be most likely leptokurtic and left-tailed for steady state cases, while both systems were typically symmetric. Lastly, the distributions were visualized using histograms and probability plots to observe any measurement tendencies, which matched the results of the other analysis steps.

7.4.3 Correlation and Linearity of the ASV and Ventilator System

Correlation was measured between the ASV and ventilator measurements. A high correlation would indicate the ASV system measures the same changes as the ventilator. However, correlation does not indicate they have a relationship. Therefore, the methods were evaluated for linearity, checked using a modified Kolmogorov-Smirnov cumulative sum test. In case they have a linear relation, the slope and intercept were found using Passing-Bablok regression. Also, the confidence interval of the slope and intercept were used to identify if the methods have significant systematic and proportional bias. The regression models were evaluated with residual plots.

Table 7.8 Pearson correlation coefficient, correlation and linearity statistical significance of the porcine lung tidal volume and dynamic compliance measurements from the ASV and ventilator systems

Experiment Case	Inspiratory Tidal Volume, *VTi*			Inspiratory Dynamic Compliance, C_{dyni}		
	Pearson Correlation Coefficient	Correlation Significant	Linearity Significant	Pearson Correlation Coefficient	Correlation Significant	Linearity Significant
1	0.19	×	×	-0.28	×	✓
2	-0.08	×	✓	0.14	×	✓
3	0.34	✓	✓	0.21	×	✓
4	0.95	✓	✓	0.85	✓	×
5	0.89	✓	×	0.61	✓	×
6	0.88	✓	✓	0.78	✓	×
Steady (1,2,3)	0.98	✓	×	-0.47	✓	×
Transient (4,5,6)	0.91	✓	✓	0.73	✓	×

The Pearson correlation coefficient was calculated for each experiment case, followed by a hypothesis test for a non-zero correlation coefficient, as seen in Table 7.8. These results found that all the individual and combined transient cases had high correlation coefficients that were statistically significant. One exception was the combined steady state case for dynamic compliance that had a negative Pearson coefficient. Also, it was observed that the dynamic compliance Pearson coefficients were lower than the tidal volume coefficients for the same cases. These results suggest that the ASV and ventilator systems are correlated and are measuring the same phenomenon. Notably, the steady state cases had low correlation coefficients and insignificant correlation, as seen in Table 7.8. This may be due to the large difference in variance between the ASV and ventilator that was measured in Chapter 7.4.2.

The modified Kolmogorov-Smirnov linearity tests found that most experiment cases were linear, as seen in Table 7.8. However, there does not seem to be any pattern to linearity, except that all the dynamic compliance transient cases were non-linear.

These results suggest only the tidal volume for case 4, 6, and the combined transient cases are linear and are strongly correlated. In general, these results are unexpected, as the tidal volume and dynamic compliance properties do not match. Interpreting the regression models and plots may explain these results.

Table 7.9 Passing-Bablok regression slope and intercept, and coefficient of determination, for steady and transient state cases for inspiratory tidal volume and dynamic compliance of the porcine lung

Parameter	Case	R^2	Slope	Slope 95% Confidence Interval		Intercept	Intercept 95% Confidence Interval	
Inspiratory Tidal Volume, *VTi* [mL]	1	-0.29	0.14	0.07	0.23	259.10	233.12	279.76
	2	0.01	0.00	-0.01	0.00	482.10	477.96	488.16
	3	-4.28	44.01	27.42	94.75	-26693.5	-58552.82	-16282.66
	4	0.87	1.86	1.70	2.00	-191.36	-267.78	-97.81
	5	0.76	1.92	1.74	2.09	-188.70	-262.33	-97.12
	6	0.72	1.83	1.60	2.07	-111.45	-235.87	-9.70
	Steady (1,2,3)	0.95	2.10	2.02	2.19	-380.10	-438.16	-333.74
	Transient (4,5,6)	0.81	1.83	1.74	1.93	-152.84	-198.62	-102.22
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	1	-0.19	0.03	-0.01	0.10	23.46	21.94	24.43
	2	-0.12	0.05	0.02	0.09	19.54	18.59	20.38
	3	-9.97	41.85	22.31	226.36	-716.90	-3982.55	-370.74
	4	0.72	1.30	1.09	1.50	1.92	-2.36	6.12
	5	0.13	1.53	1.19	1.82	-3.09	-9.95	4.01
	6	0.43	1.60	1.25	2.09	-4.95	-15.69	2.65
	Steady (1,2,3)	-0.70	0.41	0.15	0.79	14.96	7.47	20.05
	Transient (4,5,6)	0.45	1.41	1.26	1.58	-0.82	-4.26	2.72

Passing-Bablok regression was performed on the paired measurements of tidal volume and dynamic compliance for all individual and combined cases, as seen in Table 7.9. The coefficient of determination, slope, intercept and the 95% confidence intervals for the slope and intercept were measured for each regression model.

The coefficient of determinations shows that only the transient and combined case models for tidal volume reasonably fit the data. Of the other models, the individual static cases had the worst coefficient of determination that were either negative or near zero. These findings indicate a horizontal line would fit the data better than the regression models. The large difference in variance between the ASV and ventilator, measured in Chapter 7.4.2, in the static cases are likely the cause of the low coefficient of determination.

The regression model slopes and intercepts show the tidal volume and dynamic compliance measurements have the same trends. This behavior is unexpected, since dynamic compliance is

derived from tidal volume. Also, the transient case models have similar slopes and intercepts, suggesting these models are measuring the same repeatable phenomenon.

The slope and intercept confidence intervals indicate that there is systematic and proportional bias. Specifically, the slope confidence intervals never included one, and the intercept confidence intervals never included zero. The slope and intercept values are estimates of these biases, but do not explain why this relationship exists. Therefore, the regression plots were interpreted to explain the trends seen in Table 7.9.

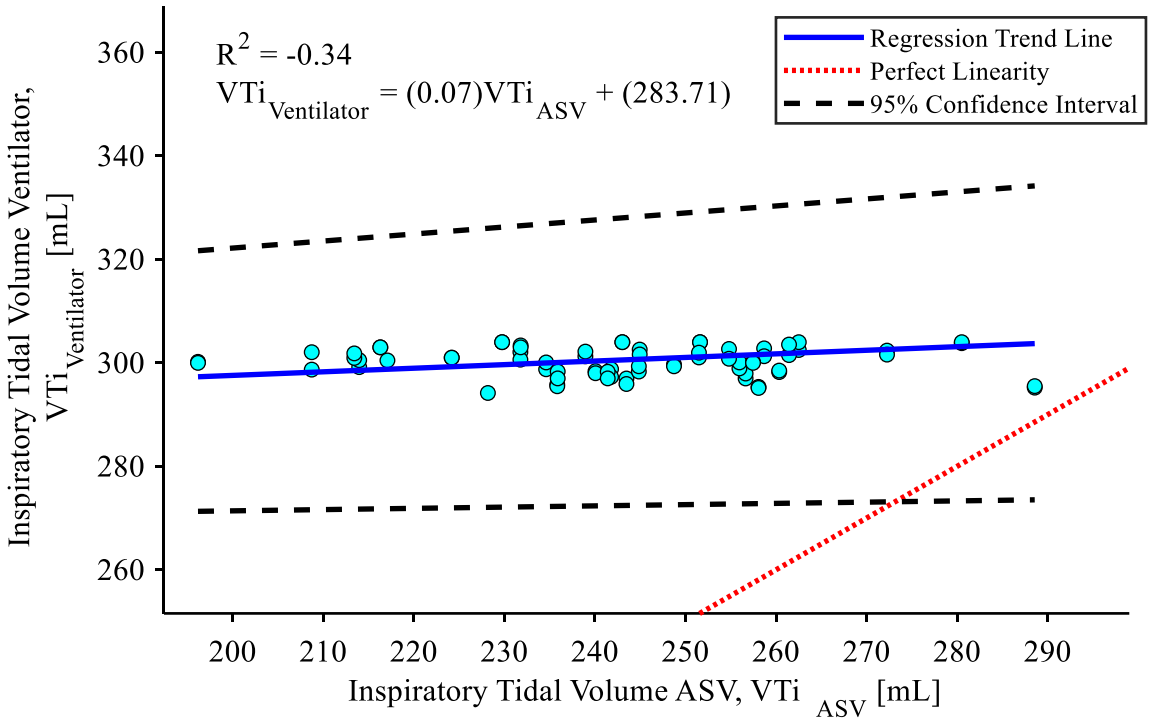


Figure 7.25 Plot of Passing-Bablok regression between the ventilator and ASV systems measurements of inspiratory tidal volume of the porcine lung during experiment case 1

The individual steady case Passing-Bablok regression plots show that these models were invalid, as seen in Figure 7.25. This occurred in Chapter 6 for the same reason, because they do not measure a wide range, and the ASV's measurement variance forms vertical lines. Passing-Bablok regression is not equipped to handle this type of data, but they had linearity because they formed a vertical line. Therefore, this data produces invalid regression models with poor Pearson correlation and coefficient of determination.

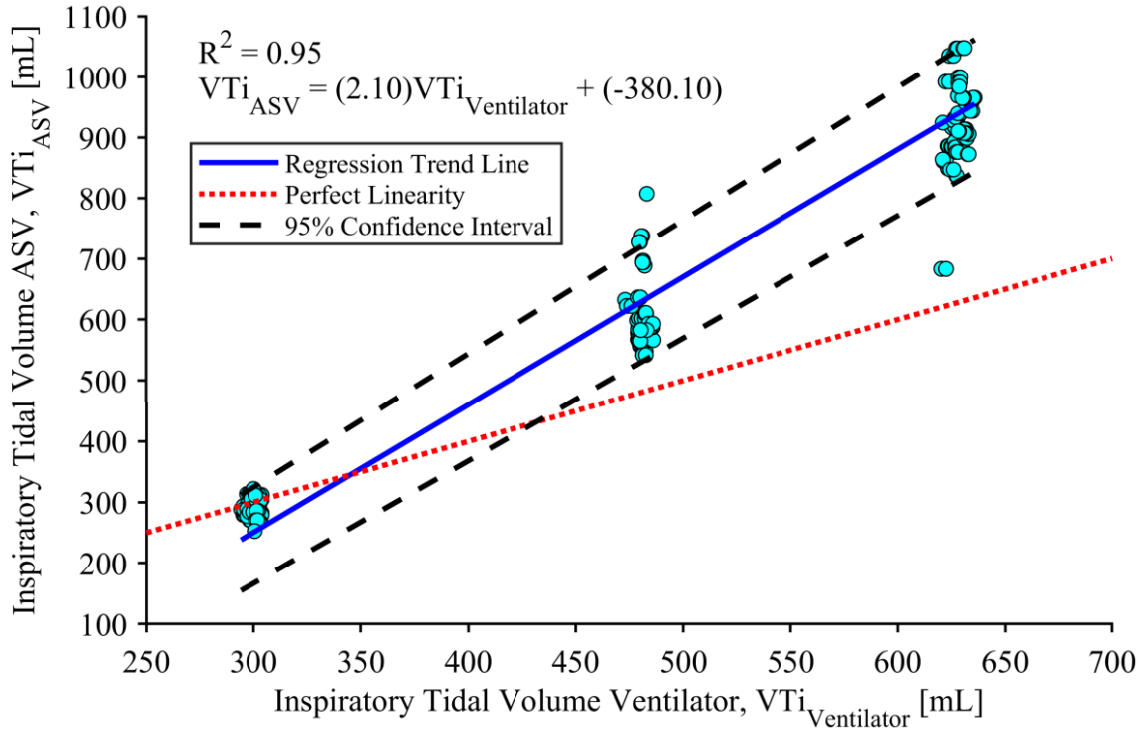


Figure 7.26 Plot of Passing-Bablok regression between the ventilator and ASV systems' measurements of inspiratory tidal volume of the porcine lung from experiment case 1, 2, and 3

The tidal volume combined steady case model had the highest Pearson correlation coefficient and coefficient of determination, despite the individual cases having poor performance. Its regression plot shows the trend line passes through the center of each cluster, from the individual cases, as shown in Figure 7.26. Despite the high dispersion along the *y*-axis, which seems to be proportional, the model is linear. Also, the slope could be lower because the trend line passes through the bottom of the first cluster. If this is the case, this model would be closer to the transient state case regression models.

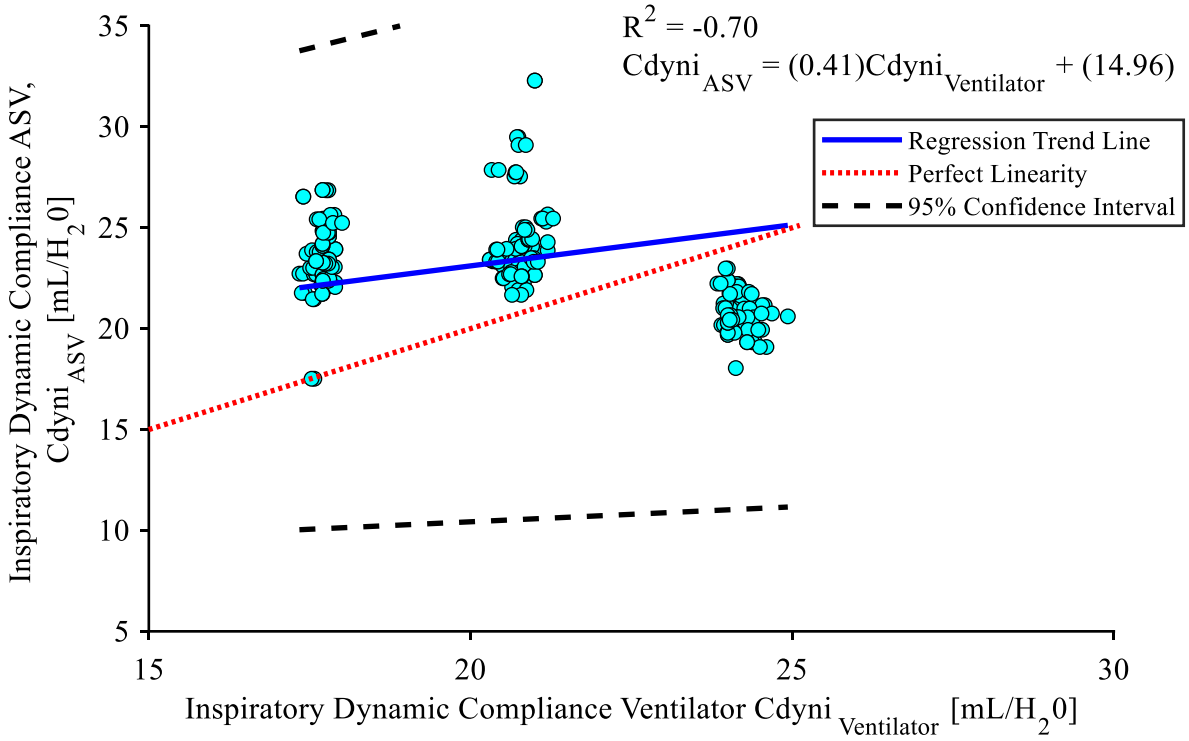


Figure 7.27 Plot of Passing-Bablok regression between ventilator and ASV inspiratory dynamic compliance of the porcine lung from experiment cases 1, 2, and 3

The dynamic compliance regression plot, shown in Figure 7.27, explains the odd behavior seen in Table 7.9. The ASV does not measure any significant change in dynamic compliance between individual steady state cases, while the ventilator does. It is likely the Passing-Bablok regression was unable to make a suitable model for this horizontal relationship that conflicts with the linear relationships seen in the transient case regression plots. It is unclear why the ASV did not measure change in dynamic compliance with the ventilator.

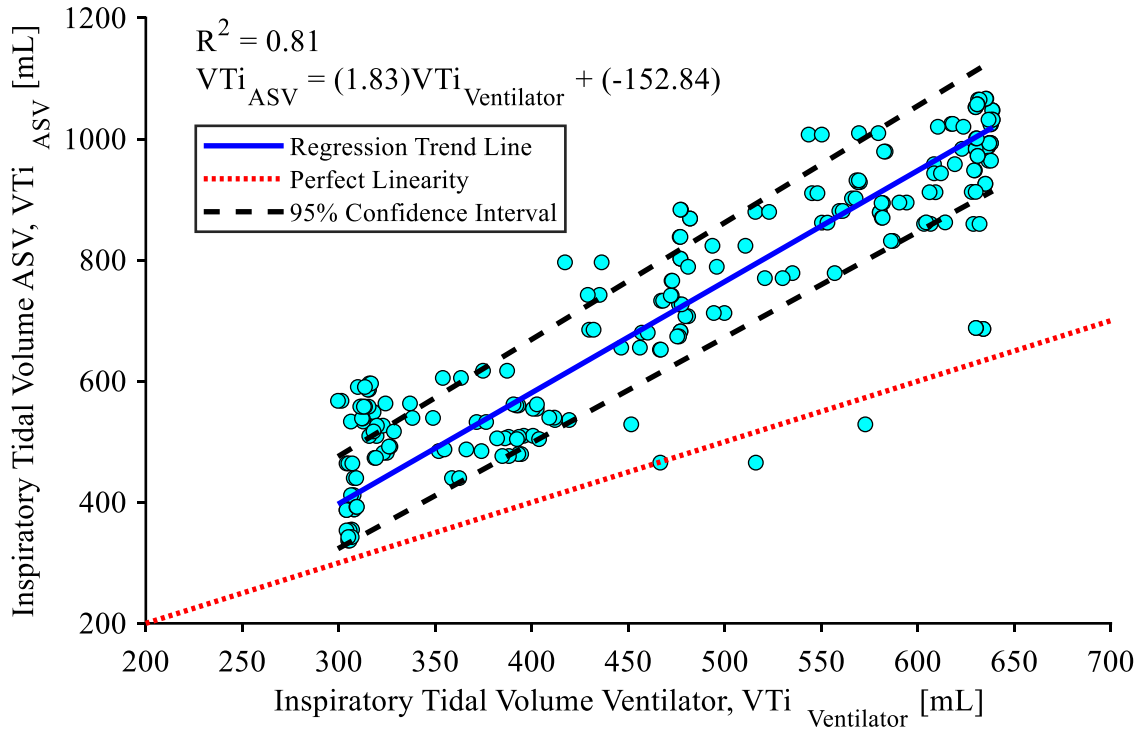


Figure 7.28. Plot of Passing-Bablok regression between the ventilator and ASV inspiratory tidal volume of the porcine lung from experiment cases 4, 5, and 6

The individual and combined transient cases have acceptable correlation and coefficient of determination in Table 7.9. The combined transient case regression plot shows that the ventilator and ASV indeed follow a linear relationship with a relatively tight dispersion, Figure 7.28. The regression model slopes and intercepts indicate systematic and proportional bias, where the ASV overestimates tidal volume relative to the ventilator.

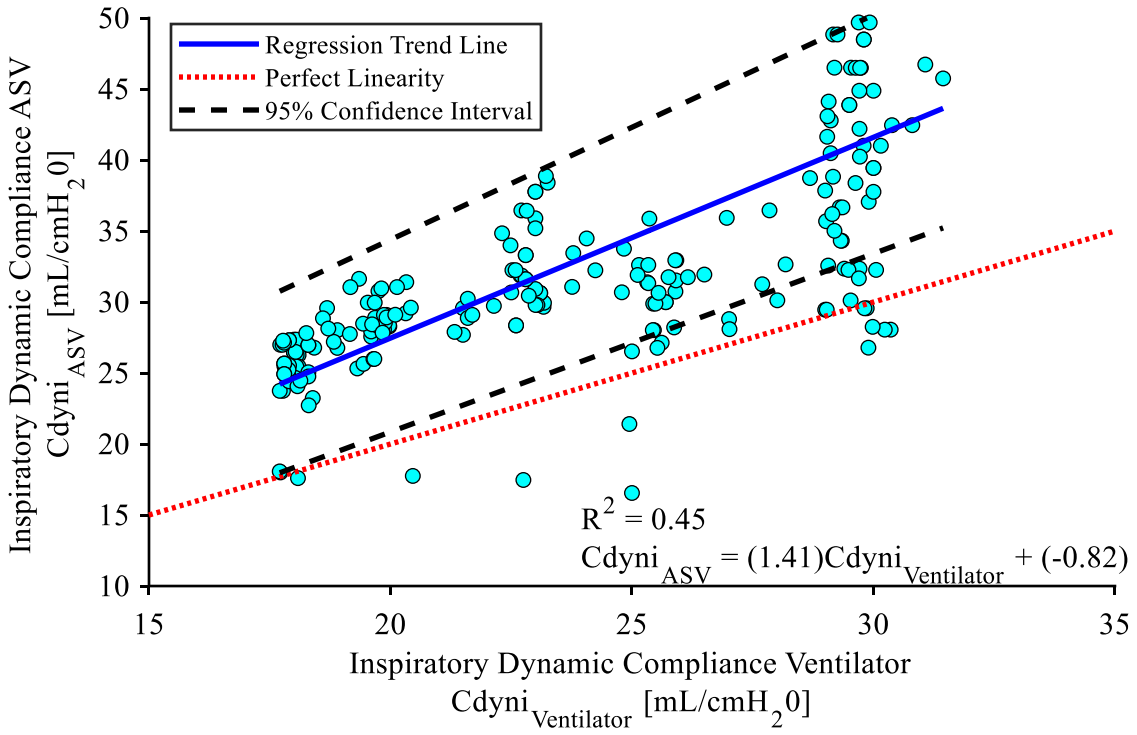


Figure 7.29 Plot of Passing-Bablok regression between the ventilator and ASV inspiratory dynamic compliance of the porcine lung from experiment case 4, 5, and 6

The regression plot for the combined transient case dynamic compliance explains why it has a poor coefficient of determination and correlation coefficient. The data points fan out from a single point to a vertical line at the minima and maxima along the *x*-axis, as seen in Figure 7.29. The other dynamic compliance regression plots have a similar proportional dispersion. Also, the datasets seem to outlier data points near and outside the 95% confidence interval that decreases the fit of the models. The regression residual plots were examined to confirm these points are outliers, and any trends, to further evaluate the fit of the models.

Table 7.10 Passing-Bablok regression residual means of the porcine lung tidal volume and dynamic compliance

Experiment Case	Inspiratory Tidal Volume, *VTi*		Inspiratory Dynamic Compliance, C_{dyni}	
	Residual [mL]	Normalized Residual [%]	Residual [mL/cmH_20]	Normalized Residual [%]
1	-0.10	-0.03	0.06	0.28
2	0.00	0.00	-0.03	-0.11
3	-11.27	-0.07	0.30	0.05
4	23.92	0.35	-0.22	-0.07
5	4.60	0.05	-2.37	-0.40
6	-27.17	-0.23	-1.42	-0.28
Steady (1,2,3)	-1.78	-0.03	-0.43	-0.15
Transient (4,5,6)	4.88	0.05	-0.94	-0.20

The residuals between the regression model predictions and the ASV measurements were used evaluated the fit of the models to the data. The residual means were measured, as seen in Table 7.10. Also, the residual means were normalized to allow comparison between the tidal volume and dynamic compliance models.

The regression models do not perfectly fit the data, based on the residual means. For most cases, the residual means were not near zero that indicates the distributions are above or below the trend lines. The only pattern shown in the table is that the steady state cases tended to have lower residual means than the transient cases. Also, based on the normalized residual means the dynamic compliance models fit the data less than the tidal volume measurements. The residual plots were examined to identify any trends to better understand residual distribution.

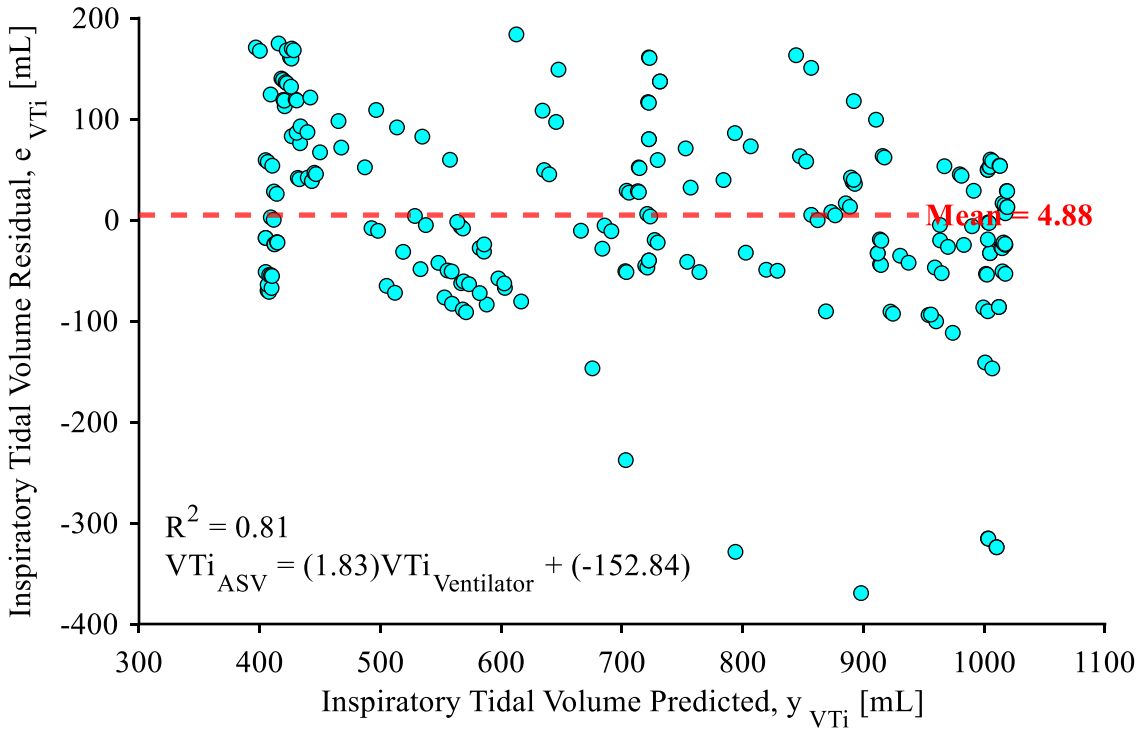


Figure 7.30 Plot of the inspiratory tidal volume Passing-Bablok regression residuals versus model predictions of the porcine lung for experiment case 4, 5, and 6

In general, the residual plots show the regression models passed through the center of the residual distributions, as seen in Figure 7.30. Also, the residual plots have outliers, points far away from the mean that are likely caused by the dispersion of the ASV measurements. Case 3 was the only exception, which had a linear trend. However, this is not significant considering that the steady state cases have invalid models because these cases do not measure performance over a range of tidal volumes.

In summary, the ASV and ventilator were found to have high correlation and linearity for tidal volume and dynamic compliance measurements for all transient state cases. Also, the regression plots indicate that the methods have significant systematic and proportional bias. These results suggest that the ASV method is measuring changes in the porcine lung with respects to ventilation. However, the ASV method is not equivalent to the ventilator's measurement system due to systematic and proportional bias.

7.4.4 **Agreement of the ASV and Ventilator System**

Agreement was measured between the ASV and ventilator measurement systems using intra-class correlation. Good agreement would suggest that the ASV system measurements are equivalent to the ventilator system. This was followed by checking if the difference in the ASV and ventilator paired measurements are normally distributed, a prerequisite for Bland-Altman analysis. The mean bias and limits of agreement from the Bland-Altman analysis were used to evaluate the agreement between the systems and if systematic bias exists.

Table 7.11 Intra-Class Correlation of the ASV and ventilator system measurements of the porcine lung

Experiment Case	Inspiratory Tidal Volume, *VTi*			Inspiratory Dynamic Compliance, C_{dyni}		
	Intra-Class Correlation	p-value	Significant ICC	Intra-Class Correlation	p-value	Significant ICC
1	0.05	0.28	×	-0.01	0.74	×
2	-1.08E-03	0.52	×	8.26E-03	0.40	×
3	1.82E-03	0.38	×	2.56E-03	0.39	×
4	0.35	0.14	×	0.34	0.14	×
5	0.41	0.12	×	0.29	0.10	×
6	0.40	0.12	×	0.41	0.10	×
Steady (1,2,3)	0.64	0.02	✓	-0.34	0.99	×
Transient (4,5,6)	0.39	0.12	×	0.34	0.12	×

Intra-class correlation was measured for each individual and combined case, followed by a hypothesis test for significance, shown in Table 7.11. These results indicate only the combined steady state case had a statistically non-zero intra-class correlation of 0.64. All the other cases had statistically insignificant intra-class correlations that were below 0.41, with most of the steady state cases having a near zero value. These results strongly suggest that the ASV and ventilator systems have poor or no agreement.

Table 7.12 Shapiro-Wilk normality test of ASV and ventilator error

Experiment Case	Inspiratory Tidal Volume, *VTi*		Inspiratory Dynamic Compliance, C_{dyni}	
	Normality	p-value	Normality	p-value
1	✓	0.89	✓	0.64
2	×	6.09E-08	×	3.43E-08
3	×	2.34E-04	×	3.24E-04
4	×	5.21E-03	×	2.91E-05
5	×	5.98E-03	×	2.90E-02
6	✓	0.08	✓	0.06
Steady (1,2,3)	×	6.98E-10	×	3.99E-08
Transient (4,5,6)	×	1.34E-03	×	2.45E-05

The normality of the difference between the paired measurements from the ASV and ventilator was found using Shapiro-Wilk hypothesis tests, as seen in Table 7.12. The tests found that the differences were normal for only case 1 and case 6, for both tidal volume and dynamic compliance. This means that it is possible the Bland-Altman analysis limits of agreement are invalid for all cases, except for case 1 and 6. Despite this, Bland-Altman analysis was performed.

Table 7.13 Bland Altman of the tidal volume and dynamic compliance measurements from the ASV and ventilator system for the porcine lung

Parameter	Experiment Case	Residual Mean	Residual Mean 95% CI		Limit of Agreement	
			Lower	Upper	Lower	Upper
Inspiratory Tidal Volume, VTi [mL]	1	-8.23	-11.01	-5.44	-36.05	19.59
	2	126.61	116.41	136.80	17.22	235.99
	3	292.03	279.81	304.24	165.04	419.02
	4	267.22	249.14	285.30	75.57	458.87
	5	223.24	203.37	243.11	0.95	445.52
	6	253.72	223.46	283.99	-4.12	511.57
	Steady (1,2,3)	140.32	126.13	154.51	-116.80	397.44
	Transient (4,5,6)	246.36	233.78	258.93	22.95	469.76
Inspiratory Dynamic Compliance, Cdyni [mL/cmH_20]	1	-3.32	-3.54	-3.10	-5.50	-1.14
	2	3.57	3.17	3.97	-0.74	7.87
	3	5.88	5.56	6.19	2.58	9.17
	4	8.60	8.00	9.19	2.29	14.91
	5	7.54	6.58	8.50	-3.23	18.31
	6	7.69	6.66	8.71	-1.05	16.42
	Steady (1,2,3)	2.21	1.76	2.67	-6.03	10.46
	Transient (4,5,6)	7.96	7.45	8.46	-16.84	0.93

The mean, the 95% confidence intervals of the means, and limits of agreement were measured from the Bland-Altman analysis for all cases, as seen in Table 7.13. All the means were found to be positive and significantly greater than zero, except for case 1 for both parameters, indicating

that the ASV was overestimating the ventilator. Also, it is possible that in the steady state cases the ASV was overestimating the ventialtor with a proportional bias because the mean increases with ventilation tidal volume for both parameters. Unsurprisingly, the mean of the combined steady case is near the average of the three individual steady state case means. On the other hand, no trend can be discerned from the mean of the transient cases and combined transient case, since they are around the same value and their true distribution mean is not known.

The means' confidence intervals suggest that systematic bias exists, since all intervals did not include zero. For all cases and parameters, the limits of agreement had a broad range, especially for the tidal volume error, which suggests the two systems are not interchangeable. Notably, the steady state cases' limit of agreement ranges increases proportionally with the ventilation tidal volume, or case number, and are smaller than in the transient cases. The Bland-Altman plots were examined to identify any trends or behaviour that explain the error distribution.

The Bland-Altman plots were evaluated to discern any trends and confirm speculation based on the mean, mean confidence intervals, and limits of agreement in Table 7.13. Overall, the Bland-Altman plots show that the ASV and ventilator do not have good agreement because they have systematic and proportional error.

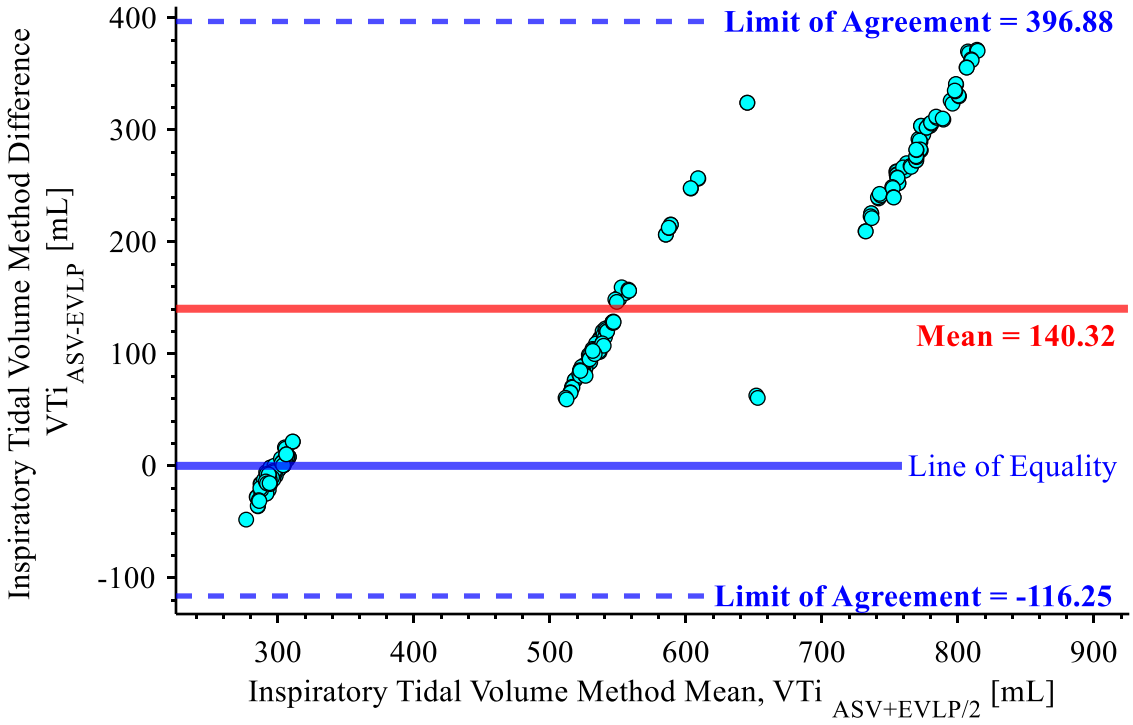


Figure 7.31 Plot of the Bland-Altman analysis of inspiratory tidal volume difference of the ASV and ventilator measurements from cases 1, 2, and 3 of the porcine lung

For the combined steady state cases each cluster is the data from one of the three steady state cases, as seen in Figure 7.31. Normally, the paired measurements of the steady state cases form a vertical line, as see in Figure 7.26. However, they get stretched out into a linear relationship in the Bland-Altman plots because their mean, which is on the *x*-axis, increases with the ASV measurements, as seen in Figure 7.31. As noticed from the means in Table 7.13, cases 2 and 3 have a large systematic error.

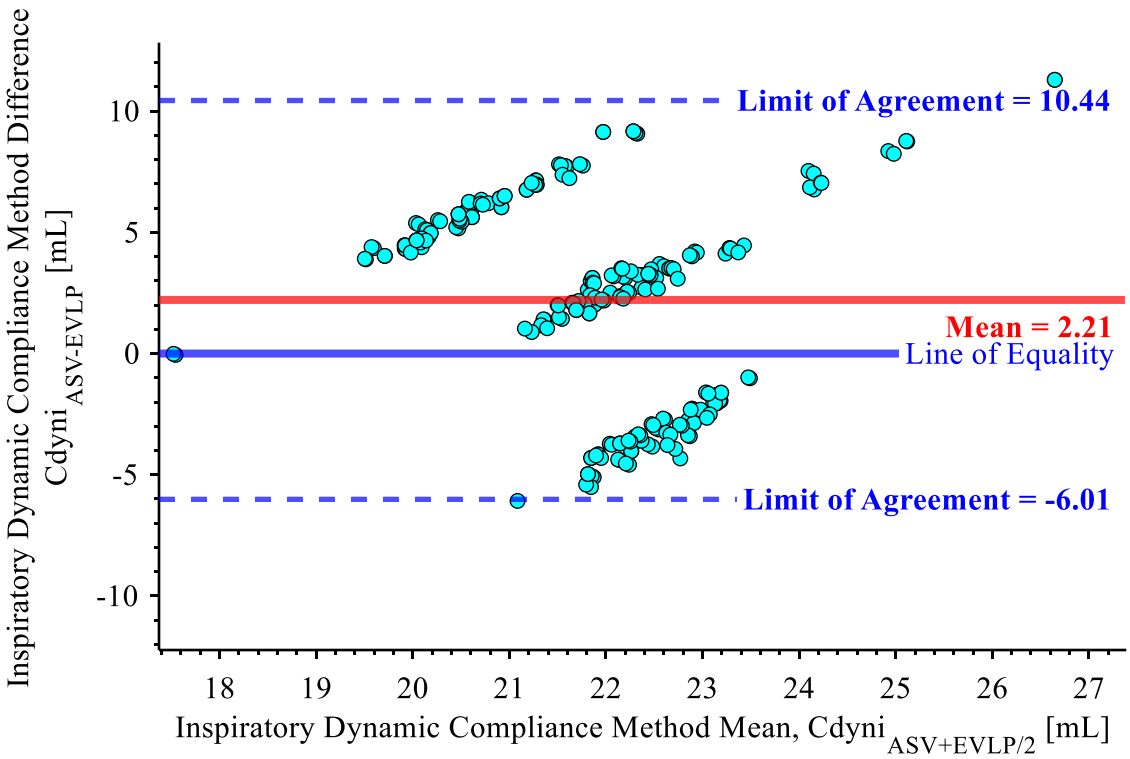


Figure 7.32 Plot of the Bland-Altman analysis of inspiratory dynamic compliance error of the ASV and ventilator system paired measurements from cases 1, 2, and 3 of the porcine lung

Similar to Figure 7.31, systematic error is shown in the Bland-Altman plot of dynamic compliance for the combined steady state cases, as seen in Figure 7.32. However, the means of each case decreases with ventilation tidal volume for dynamic compliance. This behaviour is unexpected, and its cause is unknown. However, it is hypothesized that the dynamic compliance of the porcine lung decreased with time, over the course of the experiments. Also, the linear trend for the dynamic compliance is steeper than the tidal volume. This linear trend is caused by the same reason described for Figure 7.31.

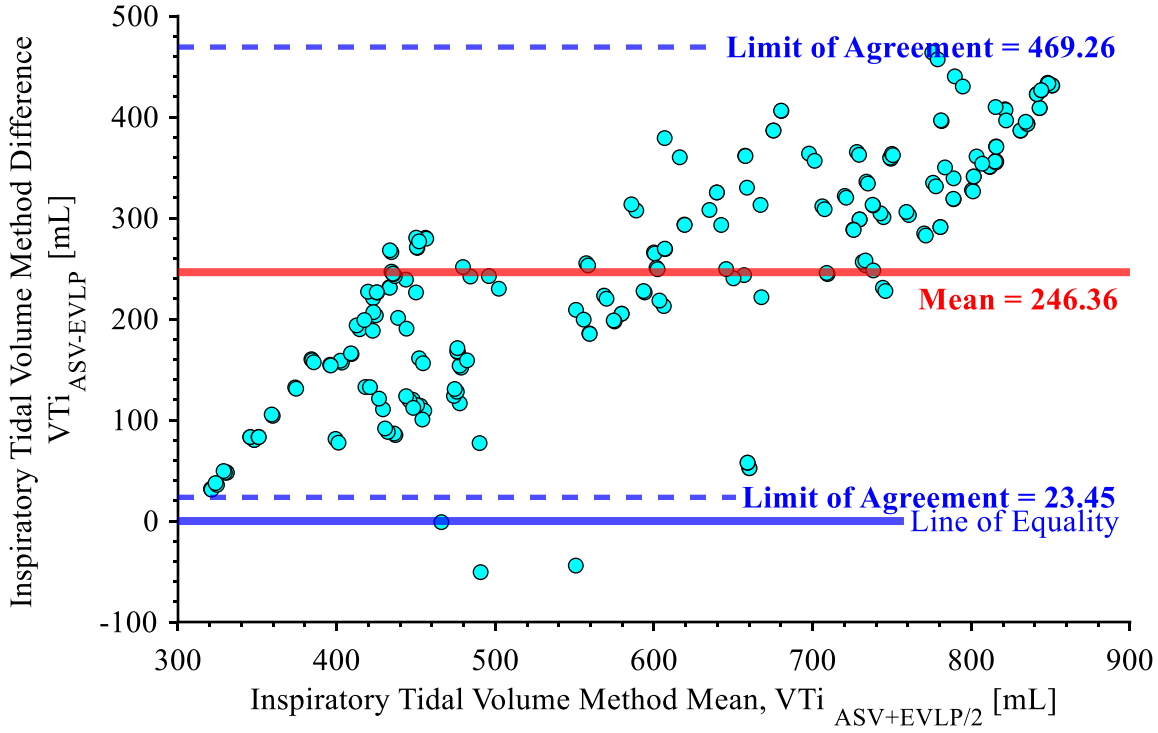


Figure 7.33 Plot of the Bland-Altman analysis of inspiratory tidal volume error of the ASV and ventilator system paired measurements from cases 4, 5, and 6 of the porcine lung

The combined transient state Bland-Altman plot shows the transient state cases have more tidal volume dispersion than the steady state cases, seen in Figure 7.33. Also, they have systematic and proportional bias since they form a linear trend line with a non-zero mean.

When the transient cases are examined separately, the cases have different distribution shapes, despite having the same ventilation range. The diagonal line of points in Figure 7.33 at 300 mL to 500 mL on the *x*-axis are from case 5. Also, the line between 750 mL and 850 mL is from case 4. Lastly, case 5 has a smaller error range and minimum than cases 4 and 6 but share the same maximum at 450 mL. Case 5 error starts at 10 mL then increases proportionally, while case 4 and 6 start at 100 mL.

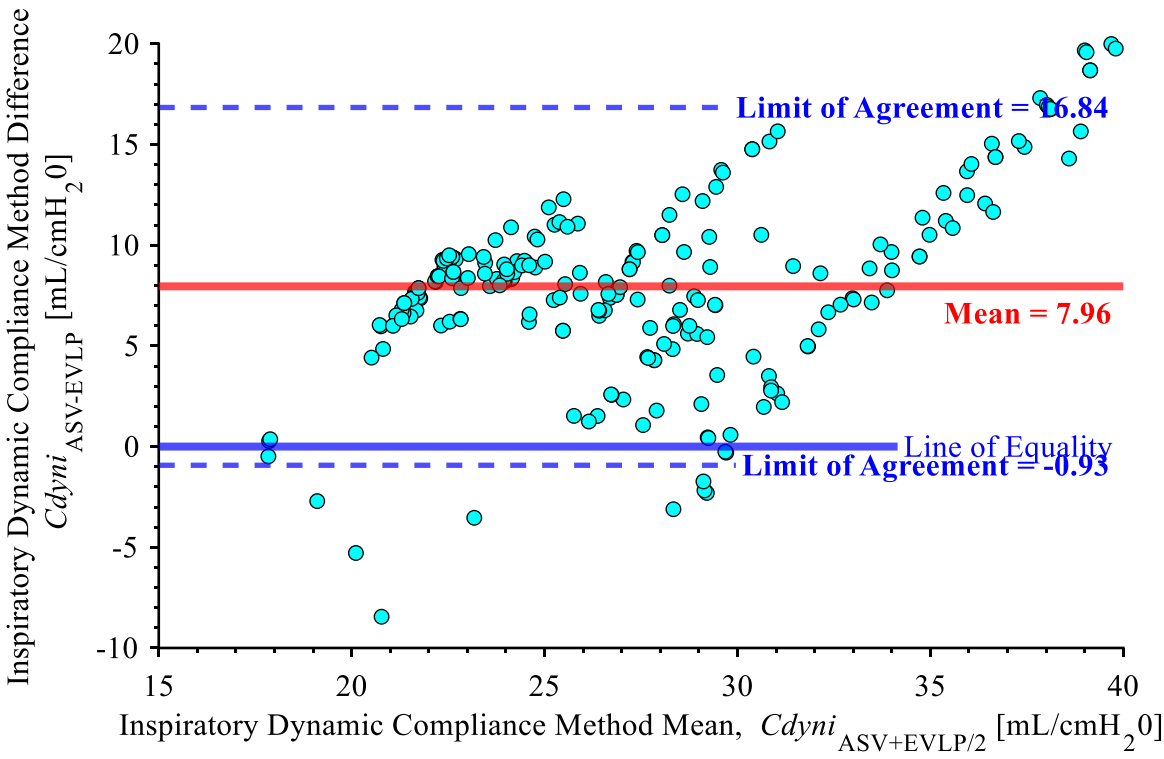


Figure 7.34 Plot of the Bland-Altman analysis of inspiratory dynamic compliance error of the ASV and ventilator system paired measurements from cases 4, 5, and 6 of the porcine lung

The dynamic compliance error of the combined transient cases does not have the same shape as the tidal volume error, as seen in Figure 7.34. Instead, the dynamic compliance error increases proportionally with the *x*-axis and disperses in both the negative and positive *y*-direction. This shape is formed by the individual transient cases where the ASV dynamic compliance did not change with the ventilator measurements. As a result, their data points form diagonal lines just like in Figure 7.31 and Figure 7.32. Lastly, there are some notable outliers at the bottom left corner of Figure 7.34.

Table 7.14 Normalized Bland Altman analysis mean and limits of agreement of the tidal volume and dynamic compliance from the ASV and ventilator of the porcine lung

Parameter	Experiment Case	Normalized Residual Mean [%]	Normalized Limit of Agreement [%]	
			Lower	Upper
Inspiratory Tidal Volume, *VTi* [mL]	1	-0.21	-0.37	-0.05
	2	0.24	0.11	0.39
	3	0.37	0.24	0.51
	4	0.41	0.23	0.59
	5	0.39	0.09	0.69
	6	0.41	0.10	0.72
	Steady (1,2,3)	0.20	-0.15	0.55
	Transient (4,5,6)	0.40	0.14	0.66
Inspiratory Dynamic Compliance, C_{dyni} [mL/cmH_20]	1	-0.33	-0.49	-0.18
	2	0.32	0.19	0.45
	3	0.28	0.14	0.42
	4	0.32	0.14	0.50
	5	0.26	-0.08	0.60
	6	0.28	-0.03	0.59
	Steady (1,2,3)	0.10	-0.27	0.48
	Transient (4,5,6)	0.28	0.04	0.56

The Bland-Altman plots were normalized by the pair mean on the *x*-axis, changing the *y*-axis from measurement units to percentage, to remove proportional error, summarized in Table 7.14. The normalized means indicate the ASV method overestimates by a significant percentage of ASV and ventilator mean. For example, the case 3 normalized mean for inspiratory tidal volume and dynamic compliance are 37% and 28%, while approximately 5% would have been acceptable. Also, the normalized limits of agreement indicate a large dispersion since the steady state cases have limits of agreement with a 30% range. The transient state and combined case limits of agreement are, as expected, large since they measure displacement across multiple tidal volume levels for ventilation.

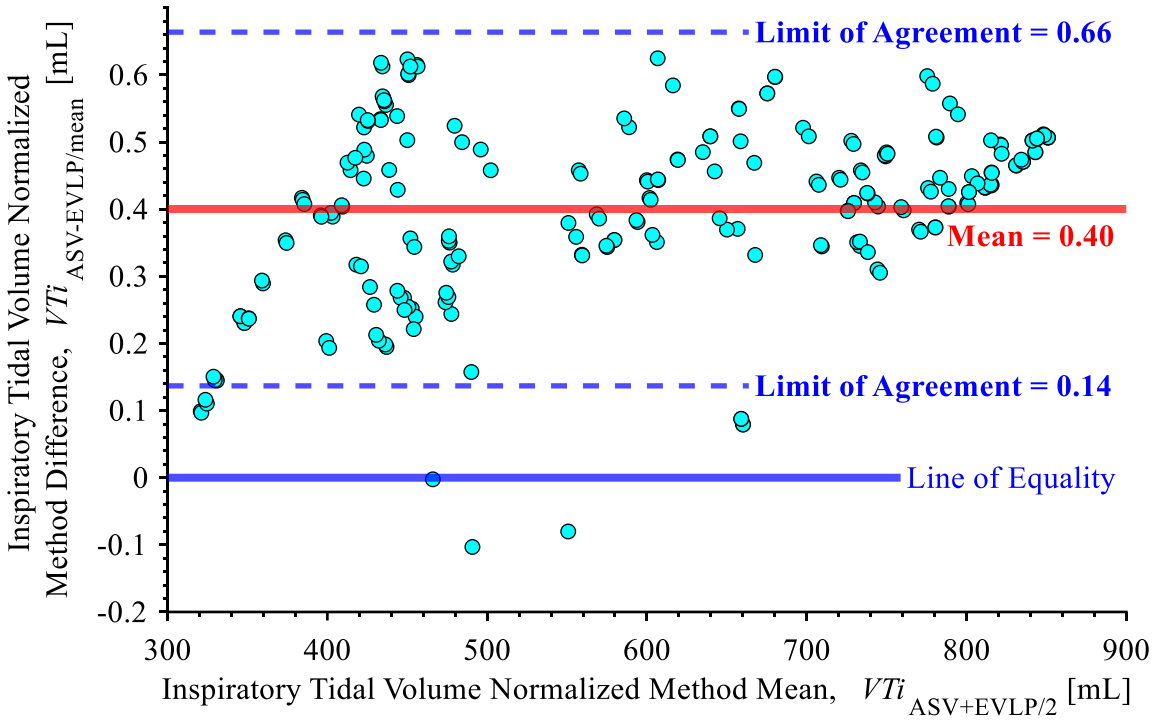


Figure 7.35 Plot of the normalized Bland-Altman analysis of the porcine lung inspiratory tidal volume of the porcine lung from the ASV and ventilator system for cases 4, 5, and 6

In general, the normalized plots just removed proportional bias seen in Figure 7.33, especially for the transient cases as seen in Figure 7.35. This made it obvious that systematic error was present, and it made it easier to identify outliers outside of the limits of agreement.

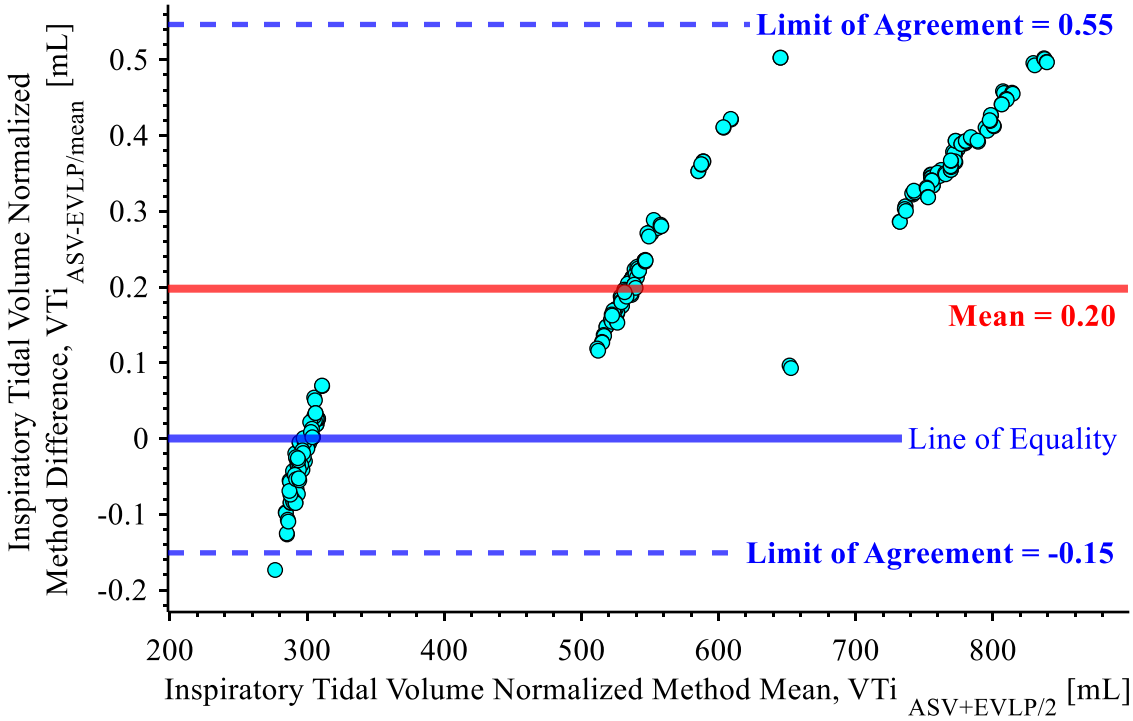


Figure 7.36 Plot of the normalized Bland-Altman analysis of the porcine lung inspiratory tidal volume of the porcine lung from the ASV and ventilator system for cases 1, 2, and 3

The combined stead state case Bland-Altman plots for both parameters are different when normalized. For the tidal volume, the linear trend lines of each cluster become steeper, as seen in Figure 7.36. Also, each cluster, or case, has a noticeably different slope that increases with the *x*-axis. For example, the cluster from case 1 is nearly vertical, while the cluster from case 3 looks diagonal in Figure 7.36.

Overall, the ASV and ventilator measurements were found to have no or poor agreement since the ASV overestimates the ventilator due to systematic and proportional error. Also, the dynamic compliance error and Bland-Altman suggests that the ASV system did not measure any meaningful change in dynamic compliance.

7.5 Porcine Lung Discussion

Measurement of the porcine lung had the same issue as the ventilator test lung in Chapter 6. For example, surface integration likely overestimated volume due to the porcine lung inflating in all directions, raising its top surface while the surface was distending. Also, the several regions of

the porcine lung were occluded in the Intel RealSense D435 images because of the porcine lung's irregular shape. As a result, surface shape information was missing and had to be interpolated to reconstruct the surface. However, the interpolated regions may not match the actual surface shape, leading to volume estimation error. Notably, the tidal volume measurements of the ASV method overestimated the ventilator. However, the ASV method observed no significant change in dynamic compliance, while the ventilator recorded a proportional increase with tidal volume.

7.6 Porcine Lung Conclusion

For six experiment cases, a porcine lung was ventilated by a Draeger Evita XL and measured by an Intel RealSense D435 to obtain depth map and color image video recordings. Surface measurements, analogous to tidal volume and dynamic compliance, were derived from these video recordings to identify localized over inflation and under inflation. Also, plethysmography measurements derived from the Intel RealSense D435 and measured by the Draeger Evita XL, were compared to evaluate the ASV method and indirectly validate the surface measurements.

The comparison discovered the ASV overestimates the ventilator, with systematic and proportional error, and has worse repeatability. Also, the two systems have poor agreement, indicating they're not interchangeable. However, their measurements have strong correlation and have statistically significant linearity, which suggests that the ASV method observes the same changes as the ventilator and that the surface measurements are valid.

8 Clinical Validation and Region Measurement

A human lung rejected for transplant was measured using the developed ASV method, providing the closest conditions to a clinical study amongst all the experiments. A processing scheme for segmenting and measuring the left and right lungs separately, was implemented, in addition to processing methods described in the previous chapters. Also, the surface interpolation method described in Chapter 7.3.3 was improved by using an alpha shape to determine the true outline of the lung. This experiment represents the finalized processing scheme developed during this research project.

8.1 Experiment Equipment

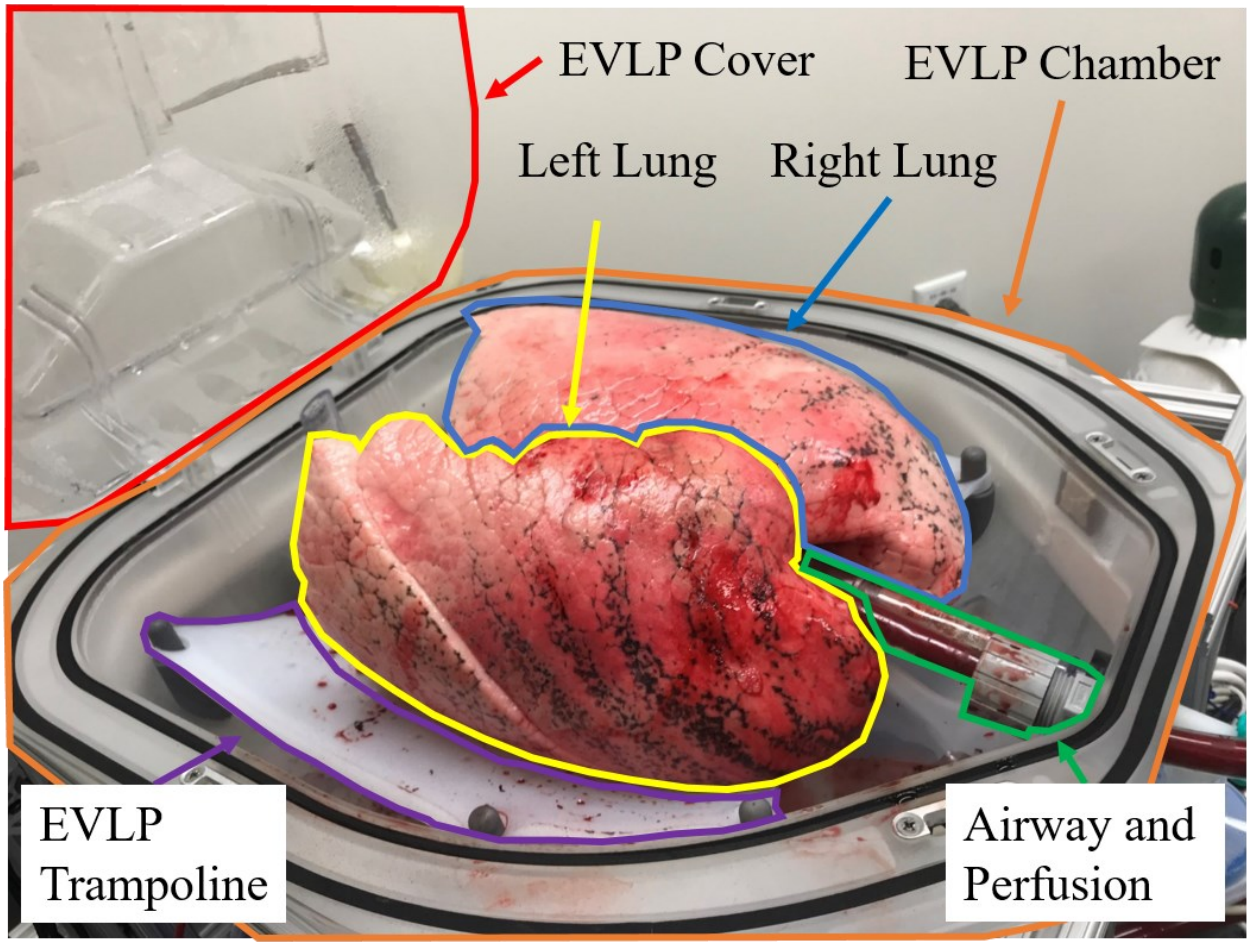


Figure 8.1 Annotated image of a human lung in an EVLP

A human lung, rejected for transplant, was acquired by Tevosol for EVLP development testing. The donor was approximately 60 kg, had a history of smoking, and other medical history

concerns that failed transplant viability checks. After 6 hours of EVLP treatment the lung was submitted for this experiment. Tevosol's experiment ended prematurely because the left lung was performing poorly. The left lung's poor condition provided an opportunity to compare the left and right lung performance using the processing scheme.

The rejected human lung was mechanically ventilated by the Draeger Evita XL using the existing airway connection between the human lung and EVLP, as seen in Figure 8.1. All equipment, except the lung and EVLP, described in Chapter 7.1 were reused for this experiment using the same procedures. Notably, the human lung's anterior surface is facing upwards in contrast to the porcine lung in Chapter 7.

8.2 Calibration and Data Acquisition

The Intel RealSense D435 was not calibrated in between the porcine lung and rejected human lung experiments. Therefore, the intrinsic and extrinsic camera parameters are the same as in Appendix D. Also, the data acquisition settings for the porcine lung experiment were re-used for the human lung using the Intel RealSense Viewer. Similarly, the Draeger Evita XL was operated using the same method as in Chapter 6.2 . However, the ventilation settings were different to achieve visually satisfactory physiological displacement.

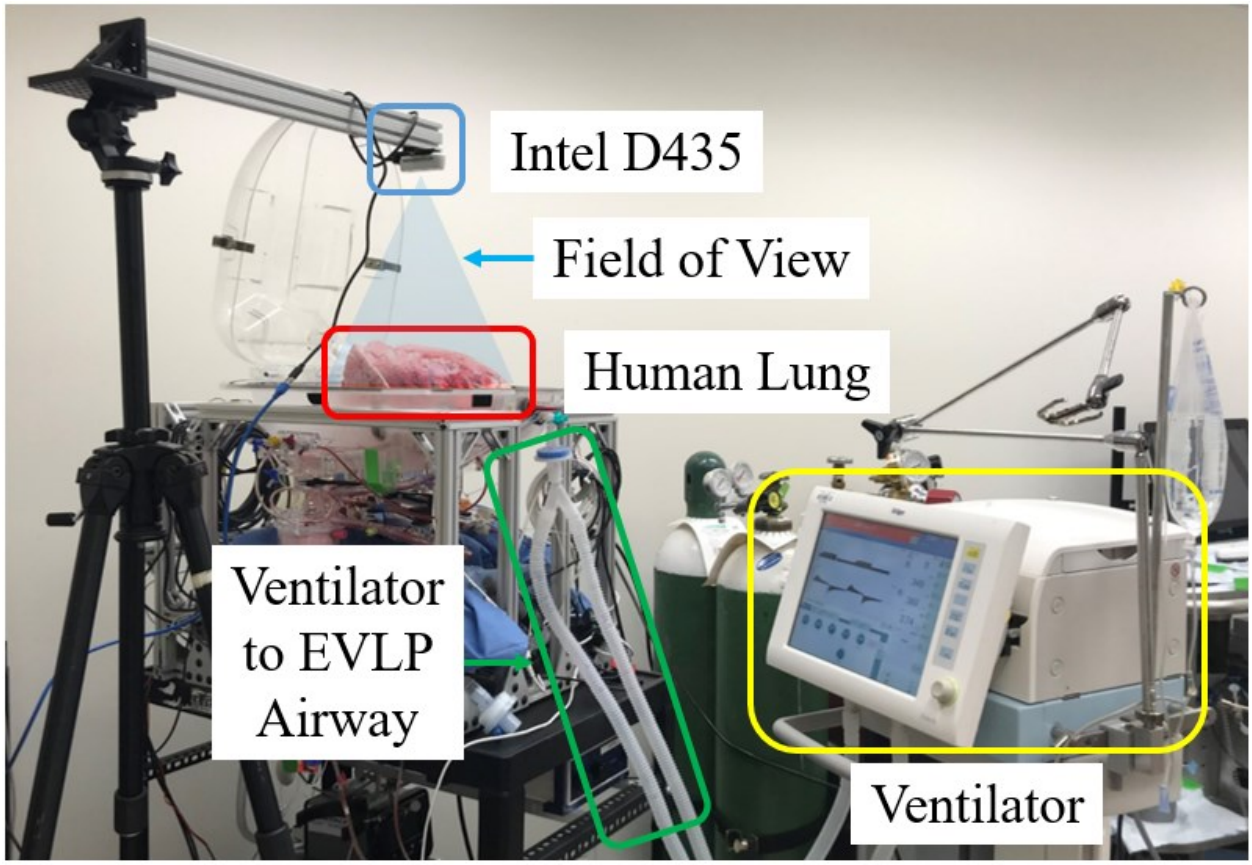
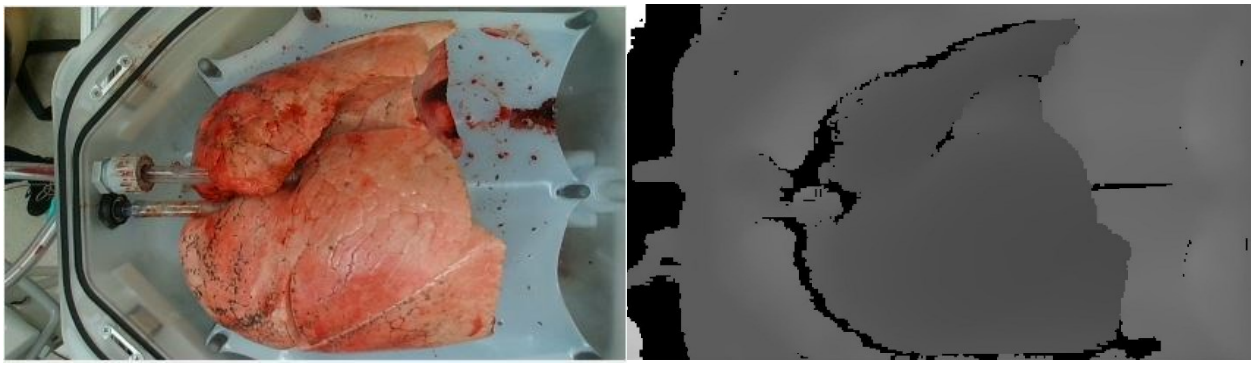


Figure 8.2 Annotated image of the human lung experimental setup

The Intel RealSense D435 was positioned above the human lung, as seen in Figure 8.2. The ASV system had a direct line of sight to the human lung, as the EVLP cover was raised. Also, the Draeger Evita XL was connected to the EVLP airway line to perform positive pressure ventilation on the human lung.



(a) *(b)*

Figure 8.3 Images of the human lung from the Intel RealSense D435 (a) color and (b) depth map streams

The color and depth map streams, as seen in Figure 8.3 (a) and (b), were recorded using the same methods and settings as described in Chapter 7.2 . The streams were recorded with a resolution of 240 × 424 pixels at 30 fps. Projective transformation was performed on the depth map to be aligned with the color image. Also, the depth maps were spatially and temporally filtered using the Intel Realsense SDK 2.0. Default ASV settings were used in the Intel Realsense Viewer application.

Table 8.1 Human lung constant ventilation settings

Parameter	**Setting**
Inspiratory Time, [s]	3.7
f, [bpm]	8.0
Slope, [-]	0.90
PEEP, [cmH20]	7
O_2, [%]	21

The same Draeger Evita XL was ventilated using the CMV auto-flow mode, just like the porcine lung experiment. However, the ventilation settings, summarized in Table 8.1, for the human lung were different than the settings for the porcine lung. The ventilation settings were adjusted to achieve visible physiological displacement of the human lung.

Table 8.2 Human lung experiment case tidal volume levels and change

Case	Experiment Type	Tidal Volume [mL/kg]	Tidal Volume [mL]
1	Steady state	8	480
2	Steady state	10	600
3	Steady state	6	360
4	Gradual Rise	6,7,8,9, and 10	360, 420, 480, 540, and 600
5	Rise	6 to 10	360 to 600
6	Gradual Fall	10,9,8,7, and 6	600, 540, 480, 420, and 360

Six experiment cases, each approximately five minutes long, were conducted for the rejected human lung. Only the tidal volume per donor weight was changed between cases. The first three cases were steady state, with a constant tidal volume of 6 mL, 8 mL, or 10 mL per kg donor. Tidal volume per donor weight was dynamically changed within Case 4, 5, and 6. Case 4 had increasing intervals of tidal volume per donor weight. Case 5 had a large step from 6 mL to 10 mL per kg donor weight. Case 6 had decreasing intervals of tidal volume per donor weight. The steady state trials are intended to provide a measurement of baseline performance, so outliers could be identified in the dynamic trials. The tidal volume changes between and within cases are tabulated in Table 8.2.

8.3 Processing Scheme

The whole rejected human lung was segmented using the interactive lazy snapping method described in Chapter 7. Also, the left and right lung were segmented individually using watershed segmentation.

8.3.1 **Human Lung Segmentation**

Figure 8.4 Image of a depth map segmented for the rejected human lung

The segmentation method described in Chapter 7 was used to segment the rejected human lung in the depth maps. Lazy snapping followed by post-processing were used to obtain binary maps of the rejected human lung from the aligned color images, as seen in Figure 8.4.

8.3.2 **Left and Right Human Lung Segmentation**

The left and right lung were segmented separately to measure their performance independently. Comparing their performance could help identify regions of failure and allows left and right asynchrony measurement. The left and right can be distinguished based on their topography. Therefore, the watershed method was used to segment the left and right lungs in the depth maps.

As described in Chapter 5, the watershed method is a region-based segmentation method. Often, the method is compared to the concept of topographical watershed where all water in a region flows to the same topographical minima, or basin. The segmentation method determines what regions, or watersheds, service each seed location acting as basins. The watershed method was implemented in three main steps:

1. Programmatically define seed regions for the left and right lung, and the background
2. Watershed segmentation of all three regions for label maps
3. Post-process the label maps for consistent results

To implement the watershed method, seed regions were automatically determined from the depth maps. Afterwards, the watershed method was used to segment the left and right lung, and the background. This process produces label maps that identify the three regions in the depth maps. Finally, the segmentation method was inconsistent, so the label maps were post-processed with spatial and temporal filtering, as well as outlier detection.

(a) *(b)*

Figure 8.5 Images of (a) the segmented and (b) the quantized depth map of the rejected human lung

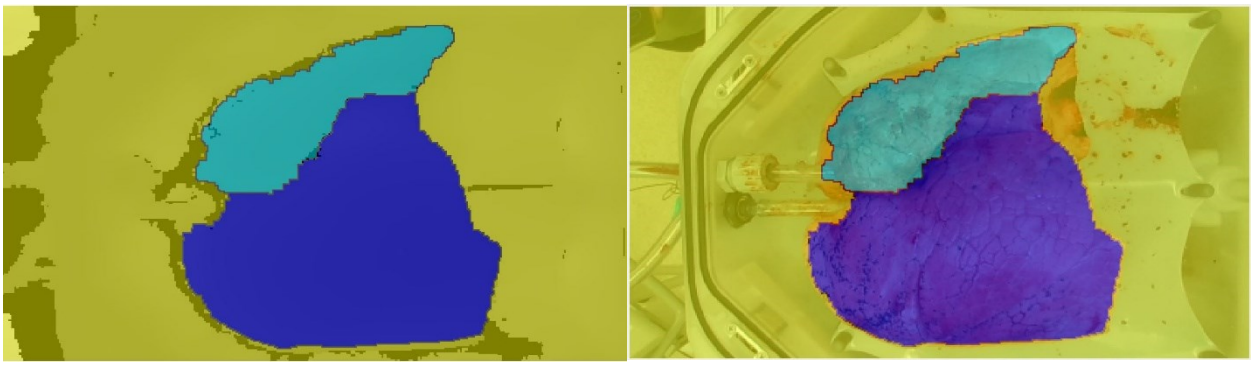
The complement of the lazy snapping binary map was taken as the background seed region. The left and right lung seed regions were obtained from the segmented depth map of the rejected human lung, as seen in Figure 8.5(a). Any depth holes in the depth map were filled using bi-linear interpolation. The segmented depth map was categorized into one of several levels, or ranges, defined by depth, as seen in Figure 8.5 (b). This step was implemented using the *imquantize()* function in MATLAB. The darkest regions in Figure 8.5 (b) are found at the top of each lung because they are the closest regions to the Intel RealSense D435 during data acquisition.

Figure 8.6 Image of the quantized and segmented depth map of the rejected human lung with the left and right lung seed regions in magenta

The minima regions were used as the seed locations for the left and right lung for watershed segmentation, as highlighted in magenta in Figure 8.6. However, through trial and error, these regions were found to be inconsistent and would provide erroneous segmentations. To solve this issue, the next depth quantization level was included as part of each seed region, or the quantization settings were adjusted. Lastly, the seed regions were morphologically eroded to ensure that they do not overlap with the background.

Figure 8.7 Image of the gradient magnitude of the segmented depth map of the rejected human lung

Watershed segmentation was implemented using the *watershed()* MATLAB function. The function segmented the gradient magnitude of the segmented depth map, as seen in Figure 8.7. All minima were removed from the gradient magnitude image, before imposing the three seed regions as new minima.

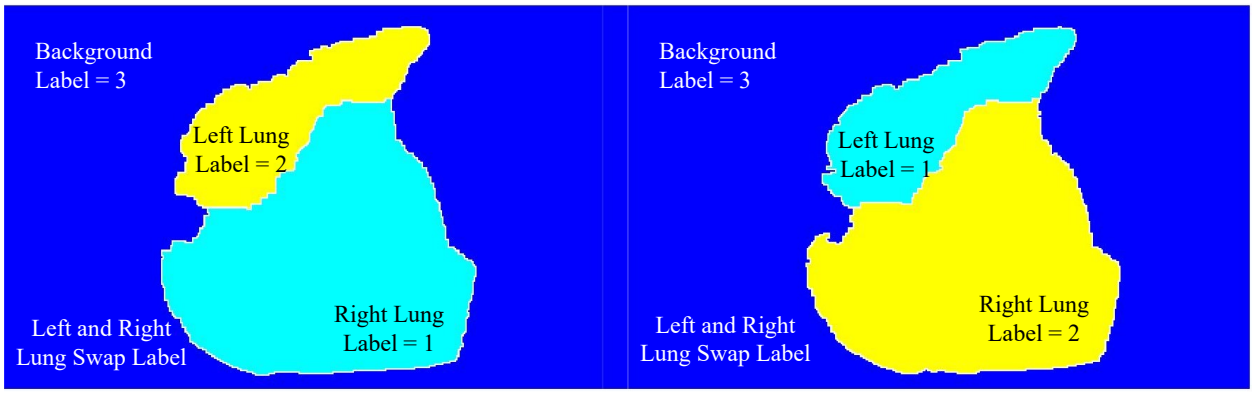


(a) *(b)*

Figure 8.8 Images of the labeled (a) depth map and (b) color image of the rejected human lung segmented for the left and right lung, and the background region

Watershed segmentation produced label maps that identify the three regions in the depth maps and color images, as seen in Figure 8.8 (a) and (b). Notably, the right lung was not fully segmented along its perimeter because of the depth holes in the depth map in that area. Also, a

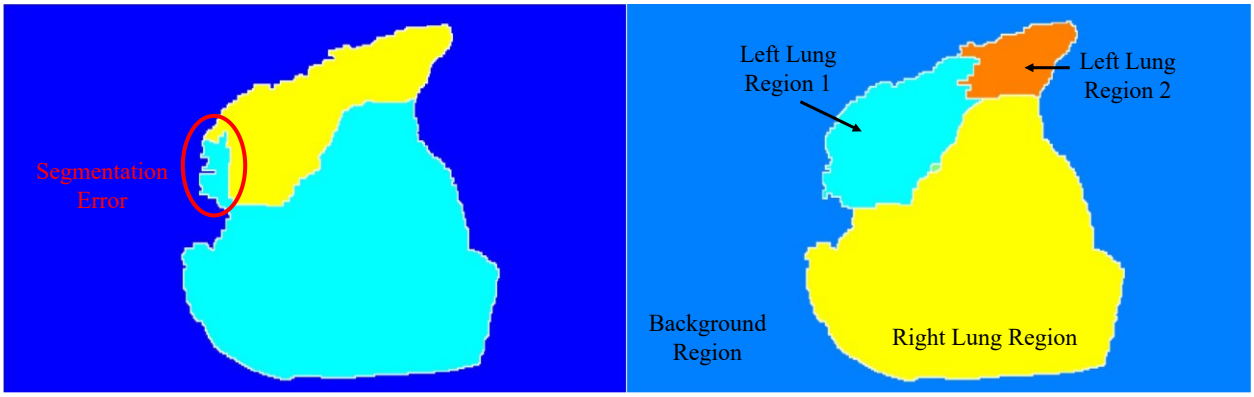
section of the right lung found at the same height as the EVLP trampoline was not segmented. This region was not segmented when using lazy snapping because it did not distend during ventilation, thus should not affect the plethysmography measurements. In general, the watershed segmentation was successful. However, there were several issues with the label maps that were corrected with post-processing.



(a) *(b)*

Figure 8.9 Annotated label maps of the left and right human lung segmentation with switching labels

The label maps use integers to identify regions. One issue with the label maps was the left and right lung swapping labelling integers between frames, as seen in Figure 8.9 (a) and (b).



(a) *(b)*

Figure 8.10 Annotated image of the label map of the left and right human lung segmentation with (a) segmentation errors and with (b) the left lung oversegmented

The watershed method sometimes had segmentation errors, where the left lung region was included in the right lung, as seen in Figure 8.10 (a). Also, the left lung sometimes was over-segmented into two different regions, as seen in Figure 8.10 (b).

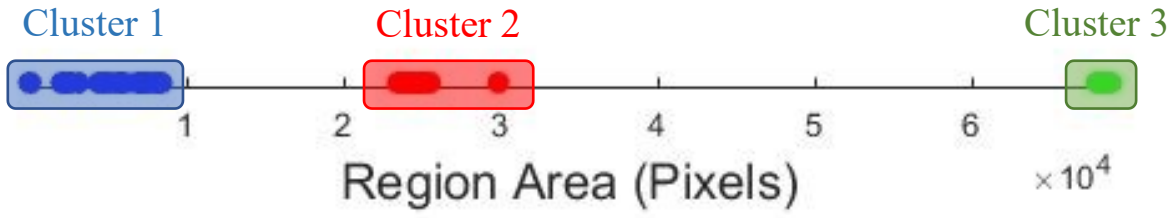


Figure 8.11 Line plot of the area of the label map regions of the human lung clustered using K-means, where K = 3 clusters

Many label map issues were resolved by segmenting regions by pixel area with K-means clustering, as seen in Figure 8.11. Clustering was performed across all regions and frames. All regions were re-assigned region labels according to their cluster.

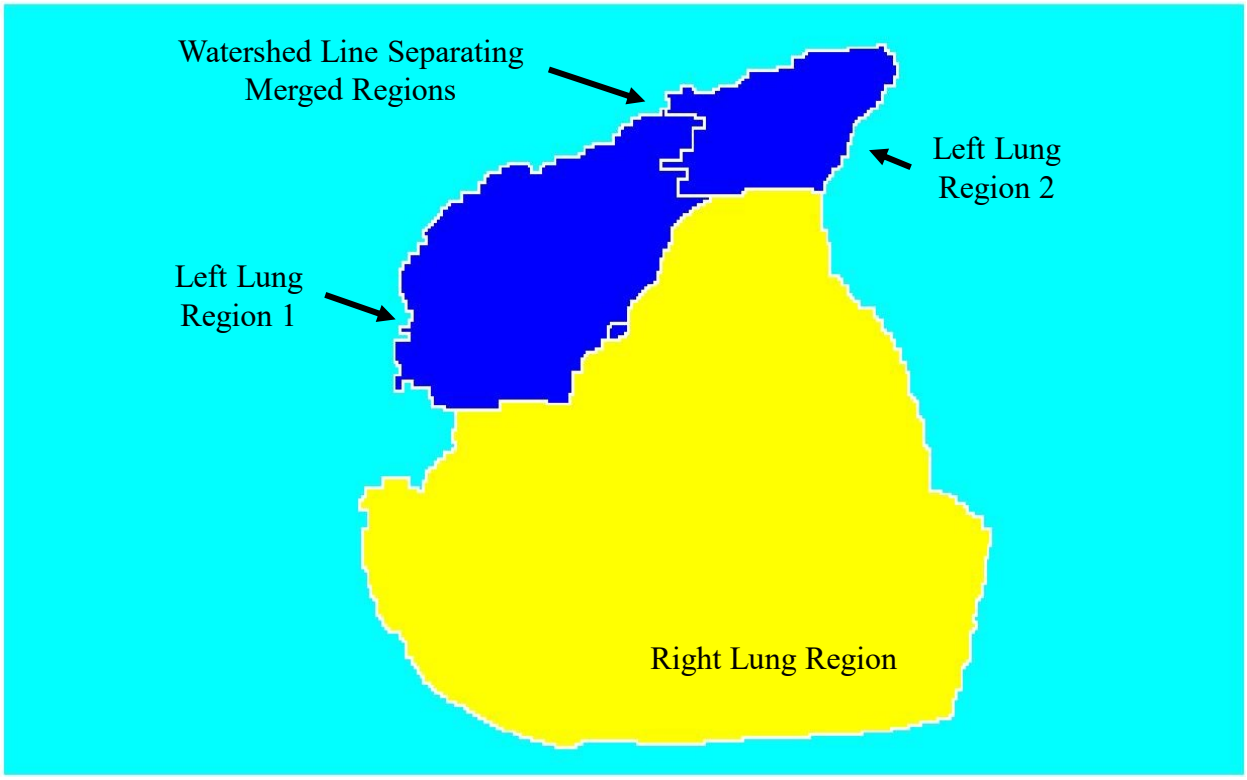


Figure 8.12 Annotated image of the label map of the left and right human lung segmentation after region merging and watershed line removal

K-means clustering had the effect of merging over-segmented regions, specifically the left lung, and obtaining consistent region labels across all frames, as seen in Figure 8.12. However, the watershed lines between the over-segmented regions and other regions remained.

Figure 8.13 Image of the label map for left and right human lung segmentation without watershed lines

The watershed lines, including between the over-segmented regions, were removed from the labelling images using a majority filter with the *modefilt()* MATLAB function. The majority filter re-assigns a pixel to the most common value within its neighbourhood. Since, all watershed lines are one pixel thick, they are always replaced, as seen in Figure 8.13.

After removing the watershed lines, the left and right lung regions were re-assigned specific label integers. The watershed method initially assigns regions a random integer label. However, this is inconvenient for a future interpolation step because it could cause the left or right lung to be interpolated as the background. Therefore, the regions in each label map were sorted by pixel area and re-assigned a label integer based on their order in size using the *sortLabels()* function in Appendix B. For example, the smallest region in each label was re-assigned the label integer "1". It was assumed the relative size of the regions, and their order in size of region, was constant, to determine the region. The last label map post-processing step was temporal filtering the left lung region, using the same method described in Chapters 6.3 and 7.3 .

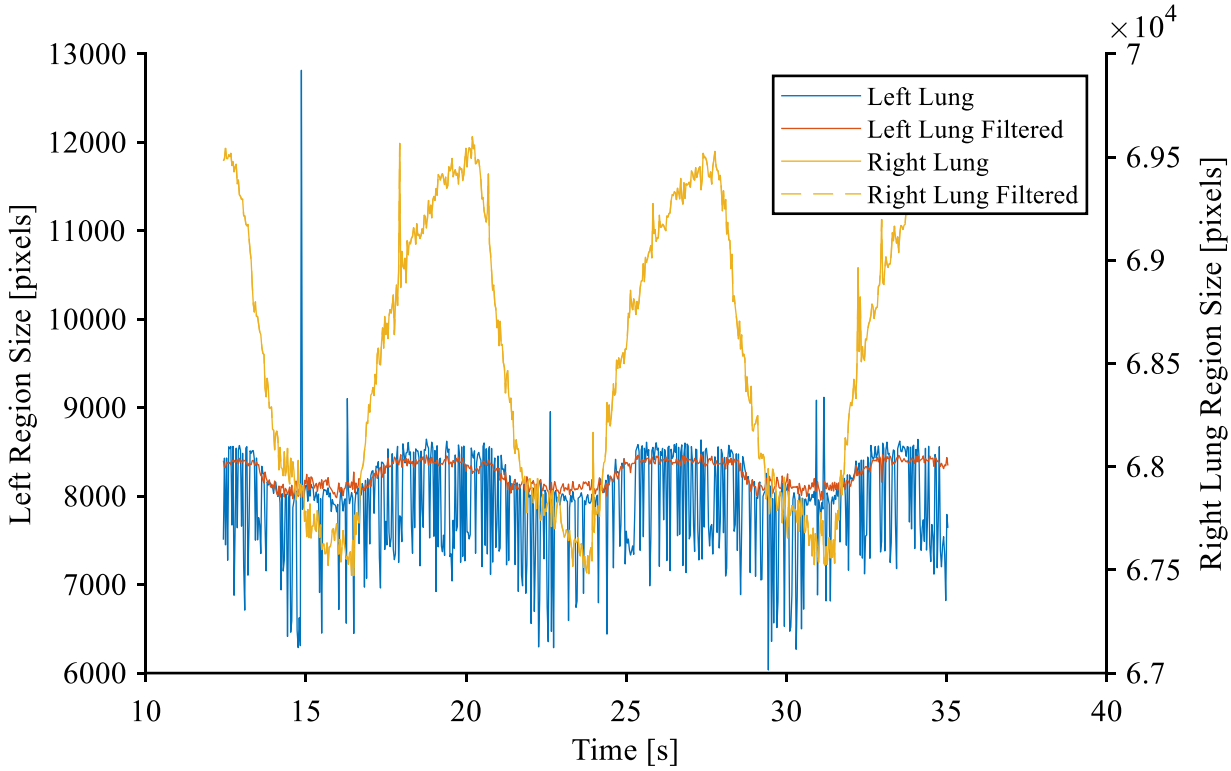
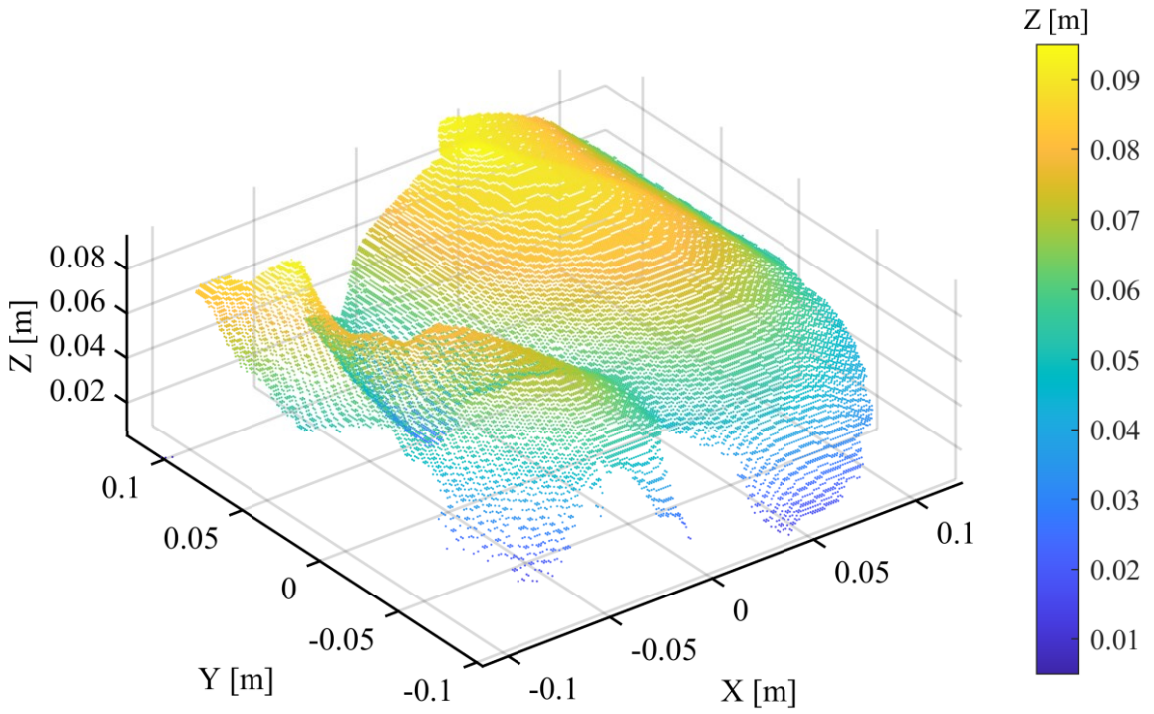


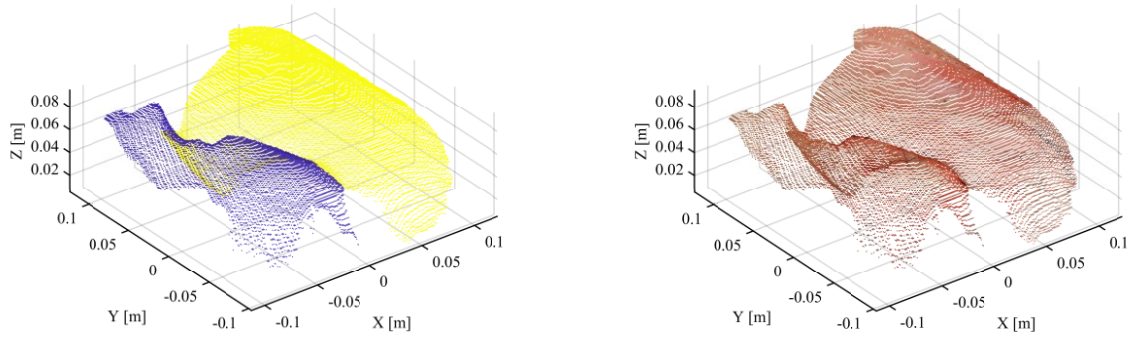
Figure 8.14. Plot of the region size of the whole lung in the label maps before and after temporal filtering

The temporal filtering significantly improved the stability of the segmentation results, especially the left lung region, as seen in Figure 8.14. The left lung region size in pixels, before temporal filtering, shows significant noise, while after filtering its size signal is denoised and has a clear periodic pattern. However, the temporal filtering of the label maps was found to not affect the right human lung region significantly.

8.3.3 **Human Lung Point Cloud Processing**



(a)



(b) *(c)*

Figure 8.15 3D plot of the porcine lung point cloud (a) colorized by depth, (b) the point cloud colorized by left and right lung label, and (c) the point cloud colorized by the color image

The segmented depth maps of the human lung were processed into point clouds, as seen in Figure 8.15 (a), using the same procedure described in Chapter 6.3.2. Also, the color and region labels were added to the point clouds, as seen in Figure 8.15 (b) and (c). The point clouds were downsampled to denoise and remove outliers and erroneous points using a box averaging filter.

8.3.4 **Surface Reconstruction of the Rejected Human Lung**

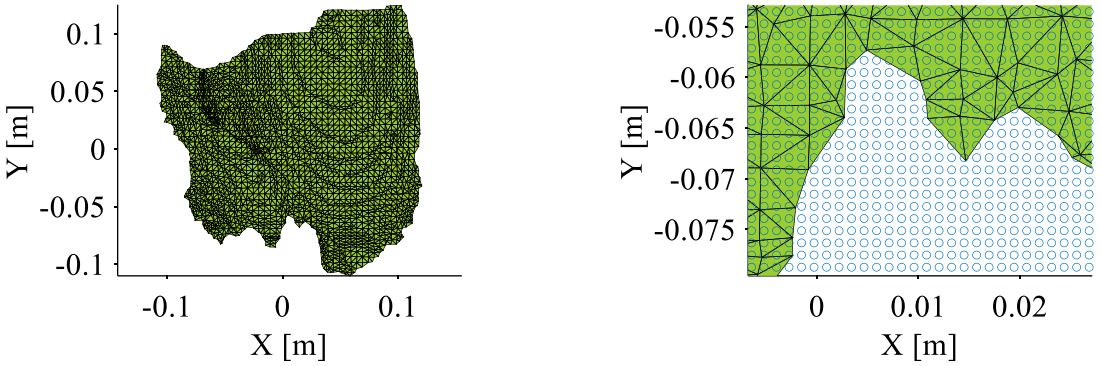


Figure 8.16 Plots of an alpha shape of the human lung (a) from a top-down view and (b) a close-up view showing which query points are within the alphas

The point clouds were interpolated to obtain surface maps of the human lung, using the same methods described in Chapter 7. However, this method has the drawback of losing the 2D outline of the lung since the scattered data interpolation fills in concave curves in the lung's outline. To preserve the lung's outline in the surface maps, an *alphaShape()* object was constructed for each frame, as seen in Figure 8.16 (a). The 2D alpha shape is formed using the x and y coordinates of the sparse point clouds. The green regions are mesh faces, which are inside the outline of the lung. The *alphaShape()* objects have a function, called *inShape()*, to determine which interpolation query points are within the alpha shape, as seen in Figure 8.16 (b). The blue circles are the interpolation query points. The query points that overlap the green alpha shape are within the outline of the lung.

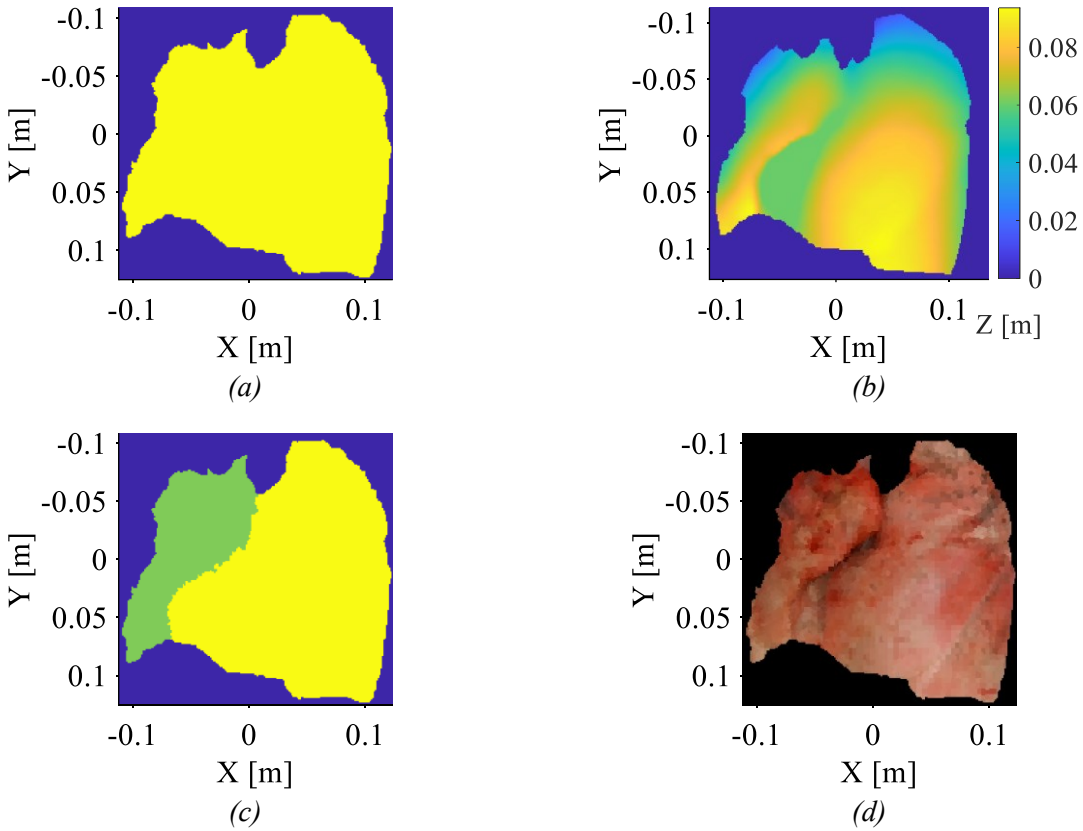


Figure 8.17 Images of the human lung (a) alpha shape binary map, (b) the surface map colorized by depth, (c) the surface label map, and (d) the human lung colored surface map

The grid points within the alpha shape were labelled, as seen in Figure 8.17 (a). The grid points within the alpha shape are marked using a binary map, visualized with respects to their x and y coordinates. This binary map can be used to identify which grid points preserve the lung's outline for any of the interpolated data types. The binary map from the alpha shape was used to filter the interpolated surface maps for depth, color, and region labelling, as seen in Figure 8.17 (b), (c), and (d).

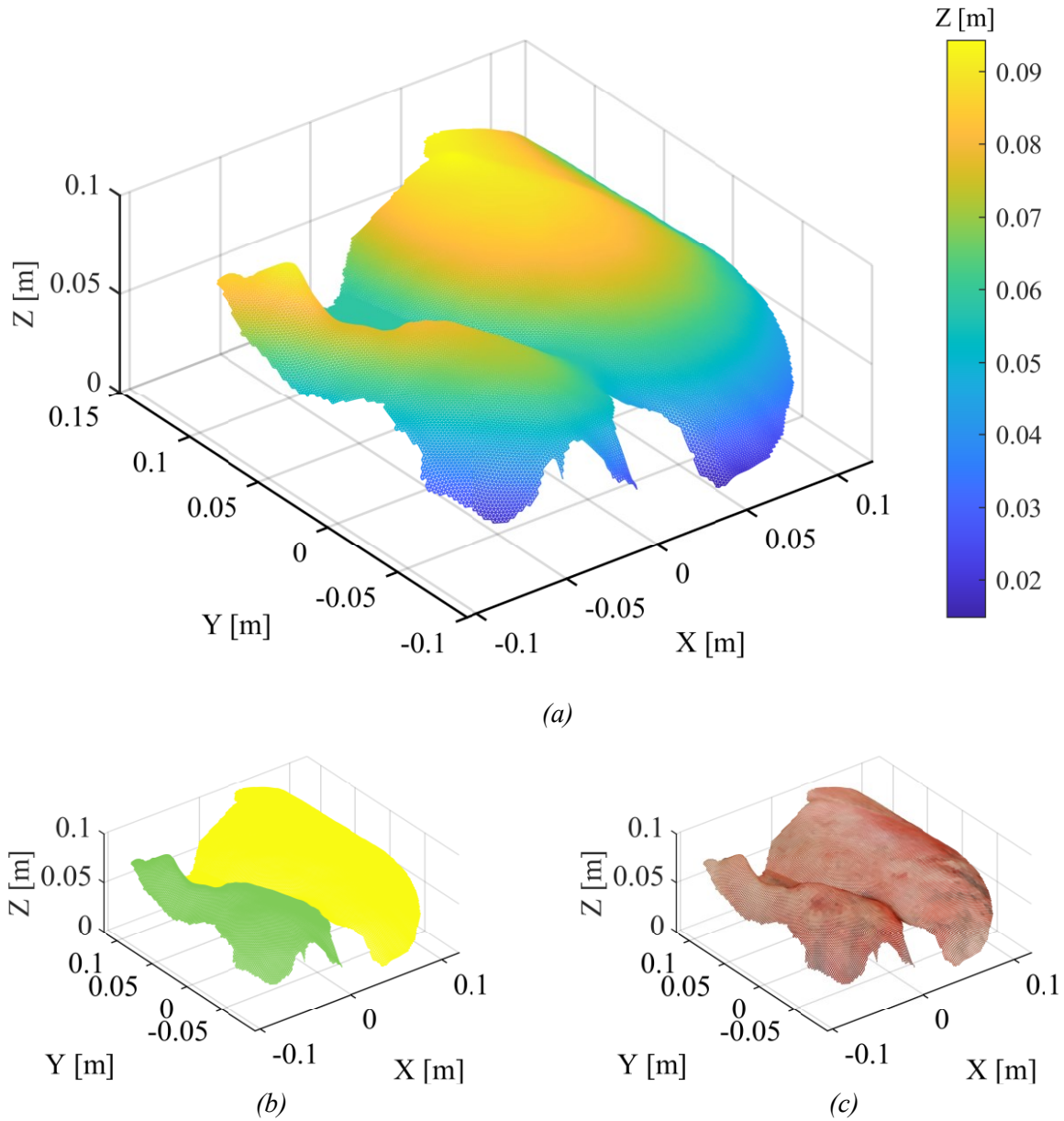


Figure 8.18. 3D plot of the human lung (a) surface map depth, (b) surface map colorized by left and right lung labels, and (c) the surface map colorized by the appearance of the human lung

The surface maps can be represented as surface meshes, as seen in Figure 8.18. As previously mentioned, the Delaunay triangulation is shared between all frames because each frame shares the same dimensions. Also, these plots were obtained using the binary maps from the alpha shape.

8.3.5 **Plethysmography Measurements of a Rejected Human Lung**

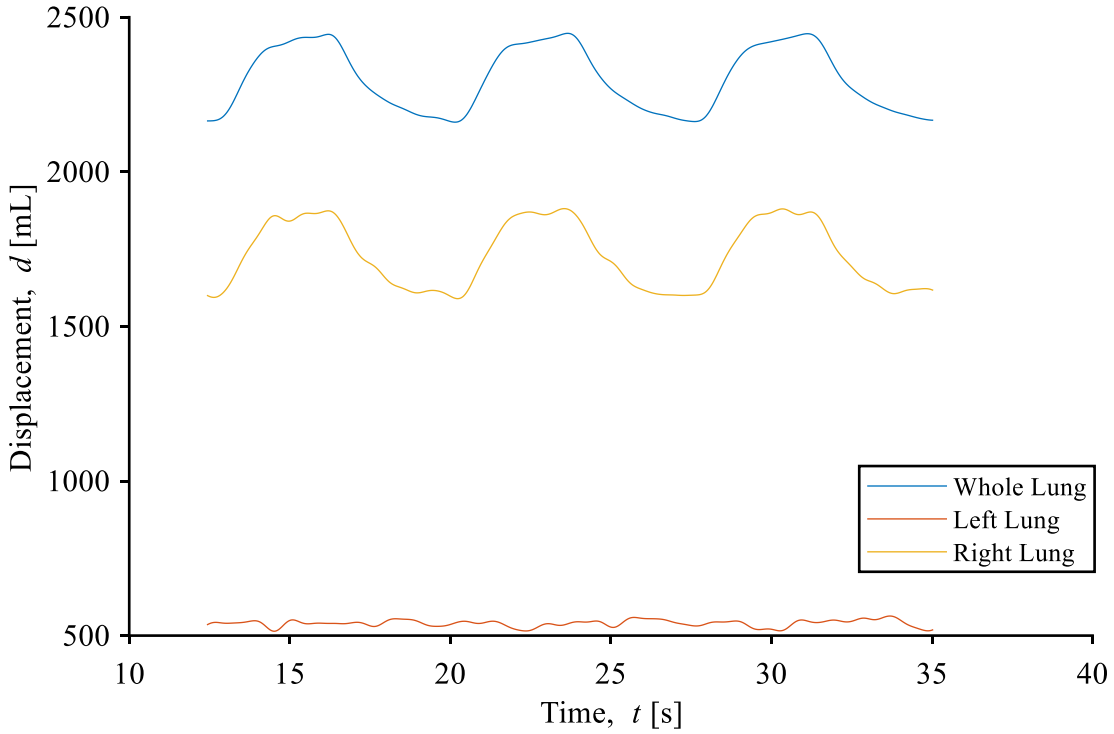


Figure 8.19. Plot of the displacement of the whole, left, and right lung of the human lung over three cycles

The surface maps were used to estimate the volume of the rejected human lung, using the previously described methods. This method was applied to the left and right lung regions separately to obtain their individual performance, as seen in Figure 8.19. The left lung did not significantly distend during ventilation, instead it was found to provide a systematic bias that summed with the cyclic displacement of the right lung to provide the whole human lung displacement. As a result, the left lung displacement signal could not be scanned for local extrema points to identify the start and end of each respiratory cycle. However, cycle segmentation was successful for the whole and right lung displacement signals. Also, plethysmography measurements were measured, such as tidal volume and inspiratory time, for the whole lung and right lung region. In addition, the cyclic average displacement was measured for the whole lung and right lung region.

Asynchrony could have been measured between the left and right lungs since their displacement signals can be compared. This metric was measured using the *estimateRegionAsynchrony()*

function in Appendix BB4. However, it was impossible to measure asynchrony since the left lung could not be segmented for respiratory cycles.

8.3.6 **Regional Measurements of a Rejected Human Lung**

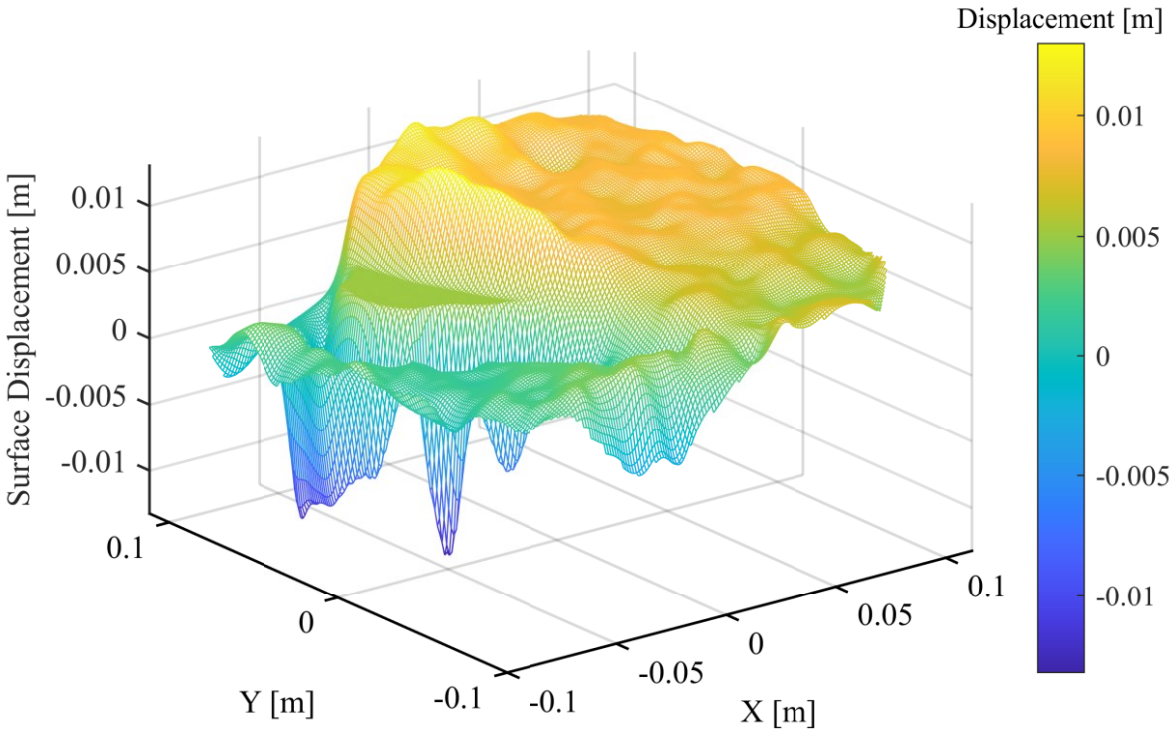


Figure 8.20. 3D plot of the surface tidal displacement of the whole human lung

Regional measurements were derived from the surface maps of the rejected human lung, as described in Chapter 7. The surface tidal displacement shows the left lung distended about 10 mm less than the right lung, as seen in Figure 8.20. Also, the plot shows the left lung has troughs along the left and right lung boundary. The surface cyclic average displacement was measured as well. However, dynamic compliance could not be found without the VSCapture pressure measurements.

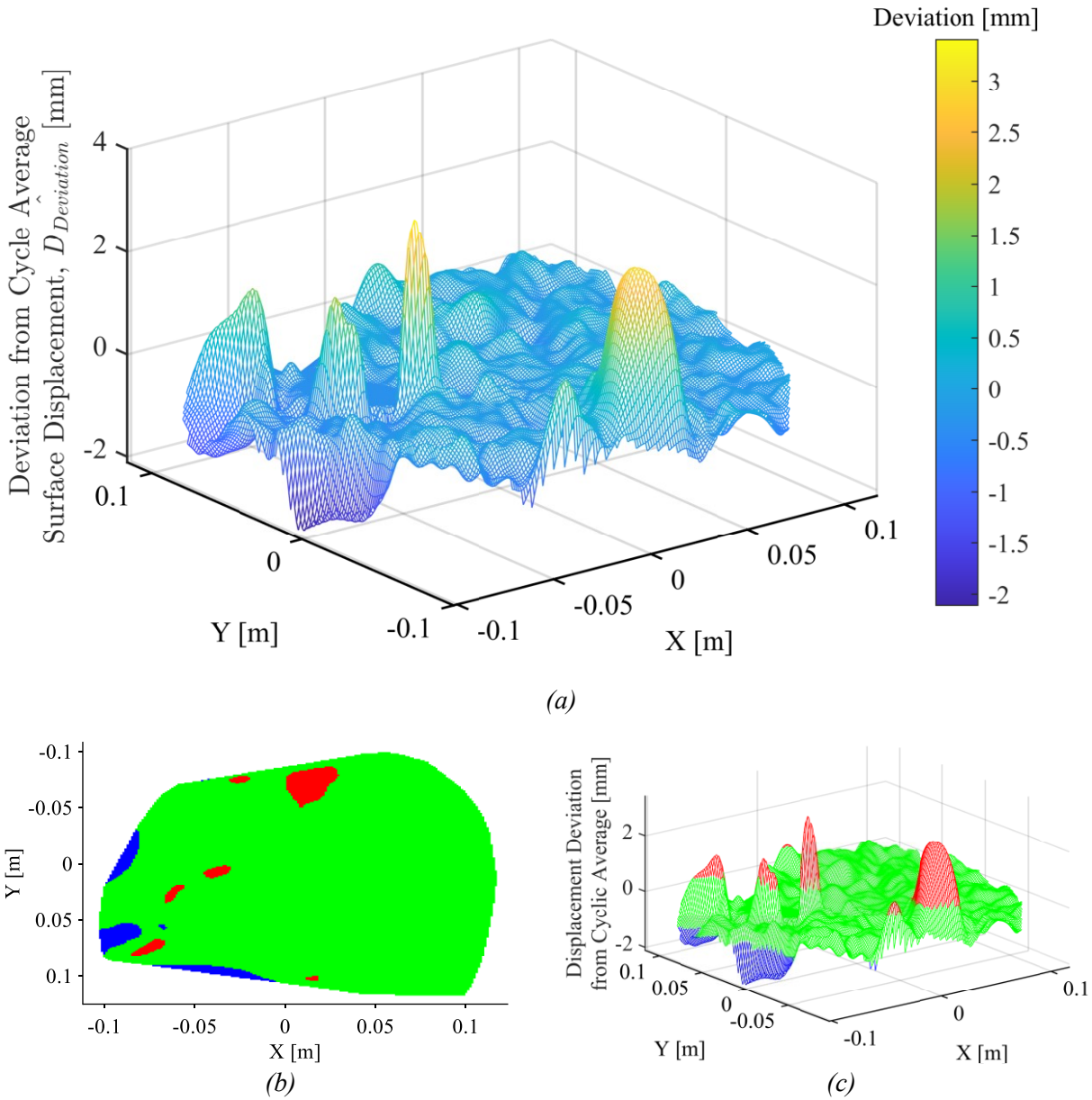


Figure 8.21. Plots of (a) the deviation from the surface cyclic average displacement of the human lung from case 1 for peak detection (b) as an image and (c) as a surface mesh where red indicates a local maxima and blue indicates a local minima outside of the deviation 95% confidence interval

The human lung was found to consistently distend during ventilation since the surface deviation from the cyclic average had a flat shape and an insignificant displacement that was at most 3 mm, as seen in Figure 8.21. Notably, there were peak regions in the surface deviation located near the front of the lung and along the left and right lung boundary. Peak detection could be used to identify the location and magnitude of these peaks. However, these peaks are inconsequential because of their small deviation in cycle average displacement.

8.4 Rejected Human Lung Discussion

The VSCapture files that record the Draeger Evita XL measurements were corrupted. As a result, method comparison was not possible for this experiment. Also, the left lung did not distend during ventilation, limiting plethysmography measurements to the whole and right lung. However, the left lung was observed not distending during this experiment, and the previous experiment with Tevosol. Additionally, Tevosol ended their experiment prematurely because the left lung failed to ventilate properly. These observations match the measured performance of the left lung determined using the ASV method. However, it is still expected that the ASV method faced the same problems encountered in Chapter 7.

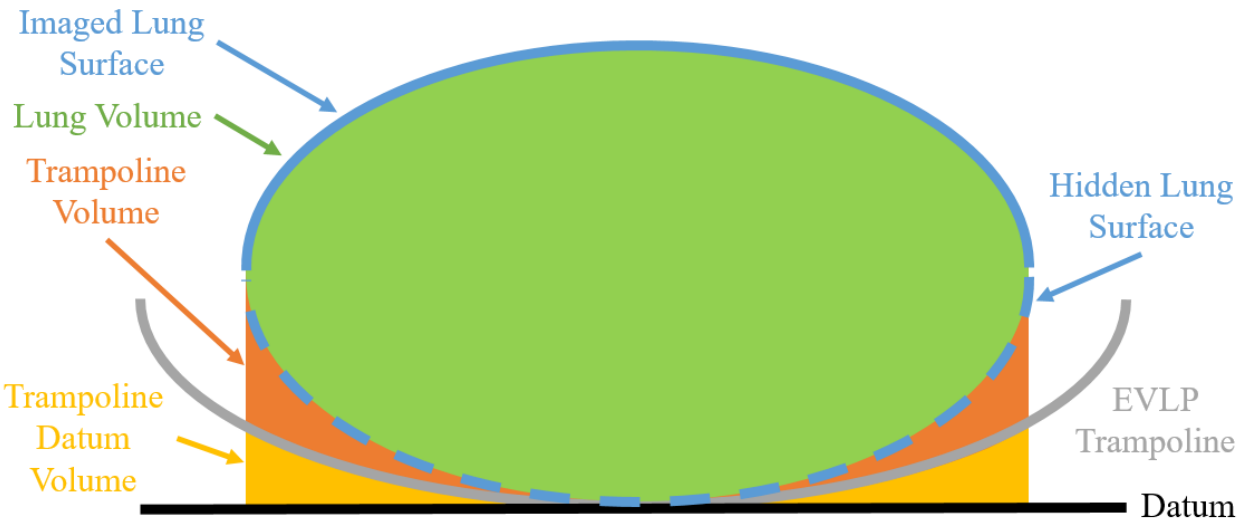


Figure 8.22 Schematic of the volume difference from surface integration with a compliant surface versus a steady state, flat plane datum

An additional issue encountered in this experiment is that the surface integration method assumes that the datum is a flat steady state plane, as seen in Figure 8.22. The human lung rests on a compliant surface that distends upward or downward during ventilation. In addition, the EVLP trampoline is not a flat surface, as its elasticity causes it to have a curved shape like the bottom of the human lung due to its weight. This invalidates the flat datum plane assumption of the surface integration method. As a result, the estimated volume is expected to have systematic and proportional error equal to the changing volume between the datum, EVLP trampoline, and the actual bottom surface of the human lung, as seen in Figure 8.22.

8.5 Rejected Human Lung Conclusion

This chapter demonstrates that the ASV method might be feasible in a clinical setting for regional measurement of donor lungs. A human lung rejected for transplant was measured using the ASV method. Plethysmography and regional measurements were obtained using surface integration and surface reconstruction. Also, the left and right lung were segmented separately to independently measure and compare their performance. However, the left lung did not significantly distend with ventilation since its displacement signal was constant. As a result, only the right lung and the whole lung displacement signals were used to derive plethysmography measurements. Also, left, and right lung asynchrony could not be measured.

Notably, Tevosol conducted an EVLP test using the same rejected human lung prior to data acquisition with the ASV method. They concluded the left lung performed poorly, which led them to prematurely end their experiment. Also, the left lung did not visibility distend during data acquisition with the Intel RealSense D435. Therefore, the ASV derived left lung measurements are likely valid since they match the results of an independent experiment and physical observations. Lastly, the ASV method results could not be compared with the ventilator because the VSCapture files were corrupted.

9 Conclusion and Future Work

9.1 Conclusion

A non-invasive active stereo vision method was developed for measuring donor lung performance during EVLP treatment to identify localized over and under inflation. The processing scheme creates an image of donor lung surface shape to measure standard plethysmography metrics, and regional metrics of tidal volume, dynamic compliance, and a respiratory cyclic average surface. The method's measurement behavior was evaluated using experimental data from a synthetic, porcine, and human lung. Also, the method was compared to spirometry-based plethysmography systems to measure correlation and agreement to assess the validity of its measurements. However, this comparison was limited to standard scalar plethysmography, as no comparable surface deformation measurements were obtained.

The method was found to correlate well with the established methods, however, has significant systematic and proportional bias resulting in poor agreement. These results suggest the active stereo vision method is measuring the respiratory cycle of the donor lung, and that the surface deformation metrics could be used to identify localized under and over inflation, and regional performance. However, the method requires further development before implementation during EVLP treatment.

9.2 Future Work

The main limitation of this study was that the regional measurements were not directly validated or compared with another method. In the future, this could be achieved with a controlled surface experiment, where the surface distends with a known geometry. Also, the main measurement limitation encountered in this study was that ASV does not track the position of points along the surface of the lungs. As encountered with the ventilator test lung experiment, the regional measurements were invalid because the entire lung moved within its respiratory cycle. Also, this limits measurement to displacement, without full field strain or stress. However, the method has several technical limitations to overcome before implementation.

The EVLP cover was raised in all experiments during this project, as imaging through the cover would introduce glare and optical warping. It is recommended that a digital or optical filter be used to mitigate glare, and a suitable ASV position be located to mitigate these effects [145]. Alternatively, a refractive stereo vision method could be used to measure through the EVLP cover [54], [146]. Also, the systematic and proportional bias in the plethysmography measurements was hypothesized to be caused by the rising of the entire donor lung along with its top surface displacement. It might be possible to correct for these biases using the developed regression models or by estimating these effects based on the donor lung's weight and size. In the porcine lung test, occlusion due to irregular surface shape left gaps in the depth map. The entire lung surface could be measured, to remove occlusion, using multiple ASVs at different angles to reconstruct a single complete surface through registration [147], [148]. Registration could be aided with fiducial markers [149] to track the position of the EVLP, and by association the donor lung. Once these limitations have been overcome, several established image processing methods could be used with the existing images to obtain other measurements.

The color and infrared images from the Intel RealSense D435 could be processed with other digital image processing methods to obtain metrics, other than surface deformation. Eulerian video magnification could be used on the color images to measure regional blood flow [45]. Also, scene flow could be used to measure the 3D displacement field of the lung's surface to measure strain [48]. With these improvements, the developed method could be a useful tool for evaluating donor lung regional performance and transplant viability.

Works Cited

[1] M. K. Shear, "State of the Art Lung Transplantation," *Dialogues Clin Neurosci*, vol. 14, no. 7, pp. 119–128, 2012.

[2] M. Valapour *et al.*, "OPTN/SRTR 2019 Annual Data Report: Lung," *Am. J. Transplant.*, vol. 21, no. S2, pp. 441–520, 2021, doi: 10.1111/ajt.16495.

[3] A. Courtwright and E. Cantu, "Evaluation and Management of the Potential Lung Donor," *Clin. Chest Med.*, vol. 38, no. 4, pp. 751–759, 2017, doi: 10.1016/j.ccm.2017.07.007.

[4] L. B. Ware *et al.*, "Assessment of lungs rejected for transplantation and implications for donor selection," *Lancet*, vol. 360, no. 9333, pp. 619–620, Aug. 2002, doi: 10.1016/S0140-6736(02)09774-X.

[5] D. Van Raemdonck *et al.*, "Lung donor selection and management," *Proc. Am. Thorac. Soc.*, vol. 6, no. 1, pp. 28–38, 2009, doi: 10.1513/pats.200808-098GO.

[6] J. Reeb, S. Keshavjee, and M. Cypel, "Expanding the lung donor pool: Advancements and emerging pathways," *Curr. Opin. Organ Transplant.*, vol. 20, no. 5, pp. 498–505, 2015, doi: 10.1097/MOT.0000000000000233.

[7] D. C. Nguyen, G. Loor, P. Carrott, and A. Shafii, "Review of donor and recipient surgical procedures in lung transplantation," *J. Thorac. Dis.*, vol. 11, no. 4, pp. S1810–S1816, 2019, doi: 10.21037/jtd.2019.06.31.

[8] A. Ali and M. Cypel, "Ex-vivo lung perfusion and ventilation: Where to from here?," *Curr. Opin. Organ Transplant.*, vol. 24, no. 3, pp. 297–304, 2019, doi: 10.1097/MOT.0000000000000647.

[9] M. T. Buchko *et al.*, "Clinical transplantation using negative pressure ventilation ex situ lung perfusion with extended criteria donor lungs," *Nat. Commun.*, vol. 11, no. 1, pp. 1–5, 2020, doi: 10.1038/s41467-020-19581-4.

[10] J. G. Y. Luc, K. Jackson, J. G. Weinkauf, D. H. Freed, and J. Nagendran, "Feasibility of Lung Transplantation From Donation After Circulatory Death Donors Following Portable Ex Vivo Lung Perfusion: A Pilot Study," *Transplant. Proc.*, vol. 49, no. 8, pp. 1885–1892, 2017, doi: 10.1016/j.transproceed.2017.04.010.

[11] T. N. Machuca and M. Cypel, "Ex vivo lung perfusion," *J. Thorac. Dis.*, vol. 6, no. 8, pp. 1054–1062, 2014, doi: 10.3978/j.issn.2072-1439.2014.07.12.

[12] M. Takahashi *et al.*, "Twenty-Four Hour Ex Vivolung Perfusion: Strategies to Stabilize Extended EVLP in a Pig Model," *J. Hear. Lung Transplant.*, vol. 37, no. 4, p. S223, 2018, doi: 10.1016/j.healun.2018.01.552.

[13] J. G. Y. Luc, K. Jackson, J. G. Weinkauf, D. H. Freed, and J. Nagendran, “Feasibility of Lung Transplantation From Donation After Circulatory Death Donors Following Portable Ex Vivo Lung Perfusion: A Pilot Study,” *Transplant. Proc.*, vol. 49, no. 8, pp. 1885–1892, 2017, doi: 10.1016/j.transproceed.2017.04.010.

[14] E. J. Charles *et al.*, “Donation After Circulatory Death Lungs Transplantable Up to Six Hours After Ex Vivo Lung Perfusion,” *Ann. Thorac. Surg.*, vol. 102, no. 6, pp. 1845–1853, 2016, doi: 10.1016/j.athoracsur.2016.06.043.

[15] G. Jiao, “Evolving Trend of EVLP: Advancements and Emerging Pathways,” *SN Compr. Clin. Med.*, vol. 1, no. 4, pp. 287–303, 2019, doi: 10.1007/s42399-019-0046-7.

[16] J. G. Y. Luc, S. J. Bozso, D. H. Freed, and J. Nagendran, “Successful repair of donation after circulatory death lungs with large pulmonary embolus using the lung organ care system for ex vivo thrombolysis and subsequent clinical transplantation,” *Transplantation*, vol. 99, no. 1, pp. e1–e2, 2015, doi: 10.1097/TP.0000000000000485.

[17] P. P. Terragni *et al.*, “Ventilatory management during normothermic ex vivo lung perfusion: Effects on clinical outcomes,” *Transplantation*, vol. 100, no. 5, pp. 1128–1135, 2016, doi: 10.1097/TP.0000000000000929.

[18] J. Retamal *et al.*, “Does Regional Lung Strain Correlate With Regional Inflammation in Acute Respiratory Distress Syndrome During Nonprotective Ventilation? An Experimental Porcine Study,” *Crit. Care Med.*, vol. 46, no. 6, pp. e591–e599, 2018, doi: 10.1097/CCM.0000000000003072.

[19] D. R. Hess, “Respiratory mechanics in mechanically ventilated patients,” *Respir. Care*, vol. 59, no. 11, pp. 1773–1794, 2014, doi: 10.4187/respcare.03410.

[20] L. Gattinoni, E. Carlesso, and P. Caironi, “Stress and strain within the lung,” *Curr. Opin. Crit. Care*, vol. 18, no. 1, pp. 42–47, 2012, doi: 10.1097/MCC.0b013e32834f17d9.

[21] J. J. Marini, “Evolving concepts for safer ventilation,” *Crit. Care*, vol. 23, no. Suppl 1, pp. 1–7, 2019, doi: 10.1186/s13054-019-2406-9.

[22] L. Gattinoni, E. Carlesso, and T. Langer, “Towards ultraprotective mechanical ventilation,” *Curr. Opin. Anaesthesiol.*, vol. 25, no. 2, pp. 141–147, 2012, doi: 10.1097/ACO.0b013e3283503125.

[23] D. Chiumello *et al.*, “Lung stress and strain during mechanical ventilation for acute respiratory distress syndrome,” *Am. J. Respir. Crit. Care Med.*, vol. 178, no. 4, pp. 346–355, 2008, doi: 10.1164/rccm.200710-1589OC.

[24] T. R. Pieber *et al.*, “Chest wall and lung volume estimation by optical reflectance motion analysis,” vol. 0585, pp. 401–408, 2008.

[25] M. Nozoe, K. Mase, and A. Tsutou, “Regional chest wall volume changes during various

breathing maneuvers in normal men," *J. Japanese Phys. Ther. Assoc.*, vol. 14, no. 1, pp. 12–18, 2011, doi: 10.1298/jjpta.Vol14_002.

[26] S. Chadha *et al.*, "Validation of respiratory inductive plethysmography using different calibration procedures," *Am. Rev. Respir. Dis.*, vol. 125, no. 6, pp. 644–649, 1982.

[27] N. Jahani, Y. Yin, E. A. Hoffman, and C. L. Lin, "Assessment of regional non-linear tissue deformation and air volume change of human lungs via image registration," *J. Biomech.*, vol. 47, no. 7, pp. 1626–1633, 2014, doi: 10.1016/j.jbiomech.2014.02.040.

[28] S. Hirsch, O. Posnansky, S. Papazoglou, T. Elgeti, J. Braun, and I. Sack, "Measurement of vibration-induced volumetric strain in the human lung," *Magn. Reson. Med.*, vol. 69, no. 3, pp. 667–674, 2013, doi: 10.1002/mrm.24294.

[29] N. Mandal, "Respirometers including spirometer, pneumotachograph and peak flow meter," *Anaesth. Intensive Care Med.*, vol. 7, no. 1, pp. 1–5, 2006, doi: 10.1383/anes.2006.7.1.1.

[30] A. Soltani, J. Lahti, K. Järvelä, J. Laurikka, V. T. Kuokkala, and M. Hokka, "Characterization of the anisotropic deformation of the right ventricle during open heart surgery," *Comput. Methods Biomech. Biomed. Engin.*, vol. 23, no. 3, pp. 103–113, 2020, doi: 10.1080/10255842.2019.1703133.

[31] P. Ferraiuoli *et al.*, "Full-field analysis of epicardial strain in an in vitro porcine heart platform," *J. Mech. Behav. Biomed. Mater.*, vol. 91, no. November 2018, pp. 294–300, 2019, doi: 10.1016/j.jmbbm.2018.11.025.

[32] W. H. De Boer *et al.*, "SLP: A zero-contact non-invasive method for pulmonary function testing," *Br. Mach. Vis. Conf. BMVC 2010 - Proc.*, pp. 1–12, 2010, doi: 10.5244/C.24.85.

[33] S. Motamedi-Fakhr, R. C. Wilson, and R. Iles, "Tidal breathing patterns derived from structured light plethysmography in COPD patients compared with healthy subjects," *Med. Devices Evid. Res.*, vol. 10, pp. 1–9, 2017, doi: 10.2147/MDER.S119868.

[34] G. Elshafie, P. Kumar, S. Motamedi-Fakhr, R. Iles, R. C. Wilson, and B. Naidu, "Measuring changes in chest wall motion after lung resection using structured light plethysmography: A feasibility study," *Interact. Cardiovasc. Thorac. Surg.*, vol. 23, no. 4, pp. 544–547, 2016, doi: 10.1093/icvts/ivw185.

[35] S. Motamedi-Fakhr *et al.*, "Evaluation of the agreement of tidal breathing parameters measured simultaneously using pneumotachography and structured light plethysmography," *Physiol. Rep.*, vol. 5, no. 3, pp. 1–16, 2017, doi: 10.14814/phy2.13124.

[36] K. Miller, *Computational biomechanics for medicine*, vol. 27, no. 3. 2011.

[37] T. S. Huang, "Computer Vision: Evolution and Promise," *Report*, 1997.

[38] S. Zhang, "High-speed 3D shape measurement with structured light methods: A review,"

Opt. Lasers Eng., vol. 106, no. December 2017, pp. 119–131, 2018, doi: 10.1016/j.optlaseng.2018.02.017.

[39] S. Giancola, M. Valenti, and R. Sala, *A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies*. 2018.

[40] E. Rundgren, "Automatic Volume Estimation of Timber from Multi-View Stereo 3D Reconstruction," 2017.

[41] Z. T. Jia and Y. F. Han, "Volume Measurement of Deposits Based on Stereo Vision and SURF," *Appl. Mech. Mater.*, vol. 743, pp. 533–536, 2015, doi: 10.4028/www.scientific.net/amm.743.533.

[42] L. Yu and B. Pan, "Full-frame, high-speed 3D shape and deformation measurements using stereo-digital image correlation and a single color high-speed camera," *Opt. Lasers Eng.*, vol. 95, no. April, pp. 17–25, 2017, doi: 10.1016/j.optlaseng.2017.03.009.

[43] M. A. Mahammed, A. I. Melhum, and F. A. Kochery, "Object Distance Measurement by Stereo Vision," *2013 Int. J. Sci. Appl. Inf. Technol.*, vol. 2, no. 2, pp. 5–8, 2013.

[44] G. Taubin, D. Moreno, and D. Lanman, *3D scanning for personal 3D printing: Build your own desktop 3D scanner*. 2014.

[45] H. Y. Wu, M. Rubinstein, E. Shih, J. Guttag, F. Durand, and W. Freeman, "Eulerian video magnification for revealing subtle changes in the world," *ACM Trans. Graph.*, vol. 31, no. 4, 2012, doi: 10.1145/2185520.2185561.

[46] X. He, R. A. Goubran, and X. P. Liu, "Wrist pulse measurement and analysis using Eulerian video magnification," *3rd IEEE EMBS Int. Conf. Biomed. Heal. Informatics, BHI 2016*, pp. 41–44, 2016, doi: 10.1109/BHI.2016.7455830.

[47] H. Lauridsen *et al.*, "Extracting physiological information in experimental biology via Eulerian video magnification," *BMC Biol.*, vol. 17, no. 1, pp. 1–26, 2019, doi: 10.1186/s12915-019-0716-7.

[48] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade, "Three-dimensional scene flow," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 3, pp. 475–480, 2005, doi: 10.1109/TPAMI.2005.63.

[49] M. Palanca, G. Tozzi, and L. Cristofolini, "The use of digital image correlation in the biomechanical area: A review," *Int. Biomech.*, vol. 3, no. 1, pp. 1–21, 2016, doi: 10.1080/23335432.2015.1117395.

[50] C. Bert, K. G. Metheany, K. Doppke, and G. T. Y. Chen, "A phantom evaluation of a stereo-vision surface imaging system for radiotherapy patient setup," *Med. Phys.*, vol. 32, no. 9, pp. 2753–2762, 2005, doi: 10.1118/1.1984263.

[51] J. J. Lee, J. H. Freeland-Graves, M. R. Pepper, W. Yu, and B. Xu, "Efficacy of thigh

volume ratios assessed via stereovision body imaging as a predictor of visceral adipose tissue measured by magnetic resonance imaging," *Am. J. Hum. Biol.*, vol. 27, no. 4, pp. 445–457, 2015, doi: 10.1002/ajhb.22663.

[52] J. J. Lee, J. H. Freeland-Graves, M. R. Pepper, M. Yao, and B. Xu, "Predictive equations for central obesity via anthropometrics, stereovision imaging and MRI in adults," *Obesity*, vol. 22, no. 3, pp. 852–862, 2014, doi: 10.1002/oby.20489.

[53] B. Tamadazte *et al.*, "Multi-View Vision System for Laparoscopy Surgery," 2014.

[54] M. Cassidy *et al.*, "Refractive Multi-view Stereo," 2020.

[55] S. Yoon, T. Choi, and S. Sull, "Depth estimation from stereo cameras through a curved transparent medium," *Pattern Recognit. Lett.*, vol. 129, pp. 101–107, 2020, doi: 10.1016/j.patrec.2019.11.012.

[56] H. Hmeidi *et al.*, "Tidal breathing parameters measured using structured light plethysmography in healthy children and those with asthma before and after bronchodilator," *Physiol. Rep.*, vol. 5, no. 5, pp. 1–12, 2017, doi: 10.14814/phy2.13168.

[57] F. Remondino and S. El-hakim, "Image-based 3D modelling: A review," *Photogramm. Rec.*, vol. 21, no. 115, pp. 269–291, 2006, doi: 10.1111/j.1477-9730.2006.00383.x.

[58] U. R. Dhond and J. K. Aggarwal, "Structure from Stereo—A Review," *IEEE Trans. Syst. Man Cybern.*, vol. 19, no. 6, pp. 1489–1510, 1989, doi: 10.1109/21.44067.

[59] S. Giancola, M. Valenti, and R. Sala, *A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies*. 2018.

[60] G. Kamberova and R. Bajcsy, "Sensor Errors and Uncertainties in Stereo Reconstruction," *Empir. Eval. Tech. Comput. Vis.*, pp. 96–116, 1998.

[61] J. Weng, P. J. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation. (Computer Vision and Image Analysis)," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 10, p. 965, 1992.

[62] G. Di Leo and A. Paolillo, "Uncertainty evaluation of camera model parameters," *Conf. Rec. - IEEE Instrum. Meas. Technol. Conf.*, pp. 598–603, 2011, doi: 10.1109/IMTC.2011.5944307.

[63] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, 2000, doi: 10.1109/34.888718.

[64] J. Weng, P. Cohen, and M. Herniou, "Camera Calibration with Distortion Models and Accuracy Evaluation." 1992.

[65] J. Salvi, X. Armangué, and J. Batlle, "A comparative review of camera calibrating methods with accuracy evaluation," *Pattern Recognit.*, vol. 35, no. 7, pp. 1617–1635,

2002, doi: 10.1016/S0031-3203(01)00126-1.

[66] W. Faig, “Calibration of Close-Range Photogrammetric Systems: Mathematical Formulation.,” *Photogramm. Eng. Remote Sensing*, vol. 41, no. 12, pp. 1479–1486, 1975.

[67] H. H. P. Wu and Y. H. Yu, “Projective rectification with reduced geometric distortion for stereo vision and stereoscopic video,” *J. Intell. Robot. Syst. Theory Appl.*, vol. 42, no. 1, pp. 71–94, 2005, doi: 10.1007/s10846-004-3023-6.

[68] A. Gruen and T. S. Huang, *Calibration and Orientation of Cameras With Computer*. 2001.

[69] Q. Wang, L. Fu, and Z. Liu, “Review on camera calibration,” *2010 Chinese Control Decis. Conf. CCDC 2010*, pp. 3354–3358, 2010, doi: 10.1109/CCDC.2010.5498574.

[70] D. V. Papadimitriou and T. J. Dennis, “Epipolar line estimation and rectification for stereo image pairs,” *IEEE Trans. Image Process.*, vol. 5, no. 4, pp. 672–676, 1996, doi: 10.1109/83.491345.

[71] C. Loop and Z. Zhang, “Computing rectifying homographies for stereo vision,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 125–131, 1999, doi: 10.1109/cvpr.1999.786928.

[72] K. Bin Lim, D. Wang, and W. L. Kee, “Virtual camera rectification with geometrical approach on single-lens stereovision using a biprism,” *J. Electron. Imaging*, vol. 21, no. 2, p. 023003, 2012, doi: 10.1117/1.jei.21.2.023003.

[73] A. Fusiello, E. Trucco, and A. Verri, “Compact algorithm for rectification of stereo pairs,” *Mach. Vis. Appl.*, vol. 12, no. 1, pp. 16–22, 2000, doi: 10.1007/s001380050120.

[74] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, “Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation,” *Proc. - 2010 Int. Conf. Embed. Comput. Syst. Archit. Model. Simulation, IC-SAMOS 2010*, pp. 93–101, 2010, doi: 10.1109/ICSAMOS.2010.5642077.

[75] R. Deriche, Z. Zhang, Q. T. Luong, and O. Faugeras, “Robust recovery of the epipolar geometry for an uncalibrated stereo rig,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 800 LNCS, pp. 567–576, 1994, doi: 10.1007/3-540-57956-7_64.

[76] K. Aggarwal, “Structure from Stereo-A Review,” vol. 19, no. 6, 1989.

[77] M. Lemmens, “A survey on stereo matching techniques,” *Int. Arch. Photogramm. Remote ...*, vol. 27, no. 1, pp. 11–23, 1988.

[78] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, “Review of stereo vision algorithms and their suitability for resource-limited systems,” *J. Real-Time Image Process.*, vol. 11, no. 1, pp. 5–25, 2016, doi: 10.1007/s11554-012-0313-2.

[79] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, “Review of stereo vision algorithms: From software to hardware,” *Int. J. Optomechatronics*, vol. 2, no. 4, pp. 435–462, 2008, doi: 10.1080/15599610802438680.

[80] A. Orriordan, T. Newe, G. Dooly, and D. Toal, “Stereo vision sensing: Review of existing systems,” *Proc. Int. Conf. Sens. Technol. ICST*, vol. 2018-Decem, pp. 178–184, 2019, doi: 10.1109/ICSensT.2018.8603605.

[81] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, “Review of stereo vision algorithms: From software to hardware,” *Int. J. Optomechatronics*, vol. 2, no. 4, pp. 435–462, 2008, doi: 10.1080/15599610802438680.

[82] A. Orriordan, T. Newe, G. Dooly, and D. Toal, “Stereo vision sensing: Review of existing systems,” *Proc. Int. Conf. Sens. Technol. ICST*, vol. 2018-Decem, pp. 178–184, 2019, doi: 10.1109/ICSensT.2018.8603605.

[83] A. Kadambi, A. Bhandari, and R. Raskar, “3D depth cameras in vision: Benefits and limitations of the hardware with an emphasis on the first-and second-generation kinect models,” *Adv. Comput. Vis. Pattern Recognit.*, vol. 67, pp. 3–26, 2014, doi: 10.1007/978-3-319-08651-4_1.

[84] D. Gallup, J. Frahm, P. Mordohai, M. Pollefeys, and C. Hill, “Variable Baseline / Resolution Stereo.”

[85] M. Kytö, M. Nuutinen, and P. Oittinen, “Method for measuring stereo camera depth accuracy based on stereoscopic vision,” *Three-Dimensional Imaging, Interact. Meas.*, vol. 7864, no. January 2011, p. 78640I, 2011, doi: 10.1117/12.872015.

[86] W. Sankowski, M. Włodarczyk, D. Kacperski, and K. Grabowski, “Estimation of measurement uncertainty in stereo vision system,” *Image Vis. Comput.*, vol. 61, pp. 70–81, 2017, doi: 10.1016/j.imavis.2017.02.005.

[87] H. Sahabi and A. Basu, “Analysis of Error in Depth Perception with Vergence and Spatially Varying Sensing,” *Comput. Vis. Image Underst.*, vol. 63, no. 3, pp. 447–461, 1996, doi: 10.1006/cviu.1996.0034.

[88] G. Di Leo and A. Paolillo, “Uncertainty evaluation of camera model parameters,” *Conf. Rec. - IEEE Instrum. Meas. Technol. Conf.*, pp. 598–603, 2011, doi: 10.1109/IMTC.2011.5944307.

[89] G. Di Leo, C. Liguori, and A. Paolillo, “Propagation of uncertainty through stereo triangulation,” *2010 IEEE Int. Instrum. Meas. Technol. Conf. I2MTC 2010 - Proc.*, pp. 12–17, 2010, doi: 10.1109/IMTC.2010.5488057.

[90] G. Di Leo, C. Liguori, and A. Paolillo, “Propagation of uncertainty through stereo triangulation,” *2010 IEEE Int. Instrum. Meas. Technol. Conf. I2MTC 2010 - Proc.*, pp. 12–17, 2010, doi: 10.1109/IMTC.2010.5488057.

[91] A. Liberati *et al.*, “Intel RealSense Stereoscopic Depth Cameras,” *Nature*, vol. 388. pp. 539–547, 2018.

[92] BDTi, “Evaluating Intel’s RealSense SDK 2.0 for 3D Computer Vision Using the RealSense D415 / D435 Depth Cameras,” no. May, pp. 1–8, 2018, [Online]. Available: https://www.bdti.com/bdti-publications-and-presentations.

[93] Intel Corporation, “Intel®RealSense - Product Family D400 Series: Datasheet,” no. June, pp. 1–121, 2020, [Online]. Available: https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf.

[94] I. Corporation, “Intel ® RealSense ™ Camera: Depth testing methodology,” *New Technol. Group, Intel Corp.*, pp. 1–18, 2018, [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/RealSense_DepthQualityTesting.pdf.

[95] Intel, “Camera depth testing methodology,” pp. 1–18, 2018, [Online]. Available: www.intel.com/design/literature.htm.

[96] I. Corporation, “Intel ® RealSense ™ Depth Module D400 Series Custom Calibration,” no. January, 2018.

[97] I. Corporation, “Intel Realsense D400 Series / SR300 Viewer: User guide,” no. May, 2018.

[98] A. Grunnet-Jepsen, J. N. Sweetser, and J. Woodfill, “Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance,” vol. 16, p. 1, 2018, [Online]. Available: https://realsense.intel.com/intel-realsense-downloads/#cal.

[99] X. Ma, X. Liu, Y. Gao, and X. Jin, “A Review on Image Segmentation Techniques,” *Jisuanji Fuzhu Sheji Yu Tuxingxue Xuebao/Journal Comput. Des. Comput. Graph.*, vol. 29, no. 10, pp. 1767–1775, 2017.

[100] R. C. Gonzalez and R. E. (Richard E. Woods, *Digital image processing*. .

[101] H. G. Kaganami and Z. Beiji, “Region-based segmentation versus edge detection,” *IIH-MSP 2009 - 2009 5th Int. Conf. Intell. Inf. Hiding Multimed. Signal Process.*, pp. 1217–1221, 2009, doi: 10.1109/IIH-MSP.2009.13.

[102] R. Muthukrishnan and M. Radha, “Edge Detection Techniques For Image Segmentation,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 6, pp. 259–267, Dec. 2011, doi: 10.5121/ijcsit.2011.3620.

[103] L. Xuan and Z. Hong, “An improved canny edge detection algorithm,” *Proc. IEEE Int. Conf. Softw. Eng. Serv. Sci. ICSESS*, vol. 2017-Novem, no. 2, pp. 275–278, 2018, doi: 10.1109/ICSESS.2017.8342913.

[104] A. A. Farag and E. J. Delp, “Edge linking by sequential search,” *Pattern Recognit.*, vol.

28, no. 5, pp. 611–633, 1995, doi: 10.1016/0031-3203(94)00131-5.

[105] A. Mathematics and S. Review, “Mathematical Morphology : A Modern Approach in Image Processing Based on Algebra and Geometry Author (s): Henk J . A . M . Heijmans Published by : Society for Industrial and Applied Mathematics Stable URL : http://www.jstor.org/stable/2132751 .,” vol. 37, no. 1, pp. 1–36, 2011.

[106] R. Unnikrishnan and M. Hebert, “Measures of Similarity.”

[107] Y. Li, J. Sun, C.-K. Tang, H.-Y. Shum, and H. Kong, “Lazy Snapping,” 2004.

[108] F. Meyer, “Topographic distance and watershed lines,” 1994.

[109] S. Beucher and C. Lantuejoul, “Use of Watersheds in Contour Detection,” *International Workshop on Image Processing: Real-time Edge and Motion Detection/Estimation*. pp. 12–21, 1979, [Online]. Available: http://www.citeulike.org/group/7252/article/4083187.

[110] F. Meyer, “The watershed concept and its use in segmentation : a brief history,” Feb. 2012, [Online]. Available: http://arxiv.org/abs/1202.0216.

[111] L. Fernandez, V. Avila, and L. Gonçalves, “A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors,” *Preprints*, no. May, pp. 1–12, 2017, doi: 10.20944/preprints201705.0170.v1.

[112] BROWN DC, “Close- range camera calibration,” *Photogramm Eng*, vol. 37, no. 8. pp. 855–866, 1971.

[113] “rs2_intrinsics coeffs[] all 0 by default,” [Online]. Available: https://github.com/IntelRealSense/librealsense/issues/1430.

[114] MATLAB, “pcdownsample.” https://www.mathworks.com/help/vision/ref/pcdownsample.html.

[115] MATLAB, “Working with Delaunay Triangulations.” https://www.mathworks.com/help/matlab/math/delaunay-triangulation.html.

[116] D. T. Lee and B. J. Schachter, “Two algorithms for constructing a Delaunay triangulation,” *Int. J. Comput. Inf. Sci.*, vol. 9, no. 3, pp. 219–242, 1980, doi: 10.1007/BF00977785.

[117] MATLAB, “Interpolating Scattered Data.” https://www.mathworks.com/help/matlab/math/interpolating-scattered-data.html.

[118] MATLAB, “scatteredInterpolant.” https://www.mathworks.com/help/matlab/ref/scatteredinterpolant.html.

[119] MATLAB, “lowpass.” https://www.mathworks.com/help/signal/ref/lowpass.html.

[120] MATLAB, “hampel.” https://www.mathworks.com/help/signal/ref/hampel.html.

[121] MATLAB, “sgolayfilt.” https://www.mathworks.com/help/signal/ref/sgolayfilt.html.

[122] MATLAB, “Taking Derivatives of a Signal.” https://www.mathworks.com/help/signal/ug/take-derivatives-of-a-signal.html.

[123] MATLAB, “medfitl2.” https://www.mathworks.com/help/images/ref/medfilt2.html.

[124] MATLAB, “imregionalmax.” https://www.mathworks.com/help/images/ref/imregionalmax.html.

[125] MATLAB, “imregionalmin.” https://www.mathworks.com/help/images/ref/imregionalmin.html.

[126] Siargo Ltd., “Model FS6122 Integrated Sensors for CPAP Applications,” Santa Clara, CA 95054, 2018.

[127] MATLAB, “designfilt.” https://www.mathworks.com/help/signal/ref/designfilt.html.

[128] MATLAB, “findpeaks.” https://www.mathworks.com/help/signal/ref/findpeaks.html.

[129] K. P. Balanda and H. L. Macgillivray, “Kurtosis: A critical review,” *Am. Stat.*, vol. 42, no. 2, pp. 111–119, 1988, doi: 10.1080/00031305.1988.10475539.

[130] P. H. Westfall, “Kurtosis as Peakedness, 1905–2014. R.I.P.,” *Am. Stat.*, vol. 68, no. 3, pp. 191–195, 2014, doi: 10.1080/00031305.2014.917055.

[131] P. Royston, “Which measures of skewness and kurtosis are best?,” *Stat. Med.*, vol. 11, no. 3, pp. 333–343, 1992, doi: 10.1002/sim.4780110306.

[132] J. Liu, W. Tang, G. Chen, Y. Lu, C. Feng, and X. M. Tu, “Correlation and agreement: overview and clarification of competing concepts and measures,” *Shanghai Arch. Psychiatry*, vol. 28, no. 2, pp. 115–120, 2016, doi: 10.11919/j.issn.1002-0829.216045.

[133] F. J. Massey, “The Kolmogorov-Smirnov Test for Goodness of Fit,” *J. Am. Stat. Assoc.*, vol. 46, no. 253, pp. 68–78, 1951, doi: 10.1080/01621459.1951.10500769.

[134] K. Rani Das, “A Brief Review of Tests for Normality,” *Am. J. Theor. Appl. Stat.*, vol. 5, no. 1, p. 5, 2016, doi: 10.11648/j.ajtas.20160501.12.

[135] J. L. Hintze, “Passing-Bablok Regression for Method Comparison,” *User’s Guid. III Regres. Curve Fitting*, pp. 1–12, 2007.

[136] “Lessons in biostatistics Comparison of methods: Passing and Bablok regression,” vol. 21, no. 3, pp. 49–52, 2011.

[137] D. Giavarina, “Understanding Bland Altman analysis,” *Biochem. Medica*, vol. 25, no. 2,

pp. 141–151, 2015, doi: 10.11613/BM.2015.015.

[138] K. O. McGraw and S. P. Wong, “Forming Inferences about Some Intraclass Correlation Coefficients,” *Psychol. Methods*, vol. 1, no. 1, pp. 30–46, 1996, doi: 10.1037/1082-989X.1.1.30.

[139] D. Liljequist, B. Elfving, and K. S. Roaldsen, *Intraclass correlation – A discussion and demonstration of basic features*, vol. 14, no. 7. 2019.

[140] P. Ranganathan, C. Pramesh, and R. Aggarwal, “Common pitfalls in statistical analysis: Measures of agreement,” *Perspect. Clin. Res.*, vol. 8, no. 4, pp. 187–191, 2017, doi: 10.4103/picr.PICR_123_17.

[141] J. M. Bland and D. G. Altman, “Measuring agreement in method comparison studies,” *Stat. Med.*, vol. 32, no. 29, pp. 5156–5171, 2013, doi: 10.1002/sim.5955.

[142] D. R. Hess, “Recruitment maneuvers and PEEP titration,” *Respir. Care*, vol. 60, no. 11, pp. 1688–1704, 2015, doi: 10.4187/respcare.04409.

[143] X. Ren and J. Malik, “Learning a classification model for segmentation,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 1, no. c, pp. 10–17, 2003, doi: 10.1109/iccv.2003.1238308.

[144] D. Stutz, A. Hermans, and B. Leibe, “Superpixels: An evaluation of the state-of-the-art,” *Comput. Vis. Image Underst.*, vol. 166, pp. 1–27, 2018, doi: 10.1016/j.cviu.2017.03.007.

[145] J. Sweetser and A. Grunnet-jepsen, “Optical Filters for Intel ® RealSense ™ Depth Cameras D400,” p. 27.

[146] J. Gedge, M. Gong, and Y. H. Yang, “Refractive epipolar geometry for underwater stereo matching,” *Proc. - 2011 Can. Conf. Comput. Robot Vision, CRV 2011*, pp. 146–152, 2011, doi: 10.1109/CRV.2011.26.

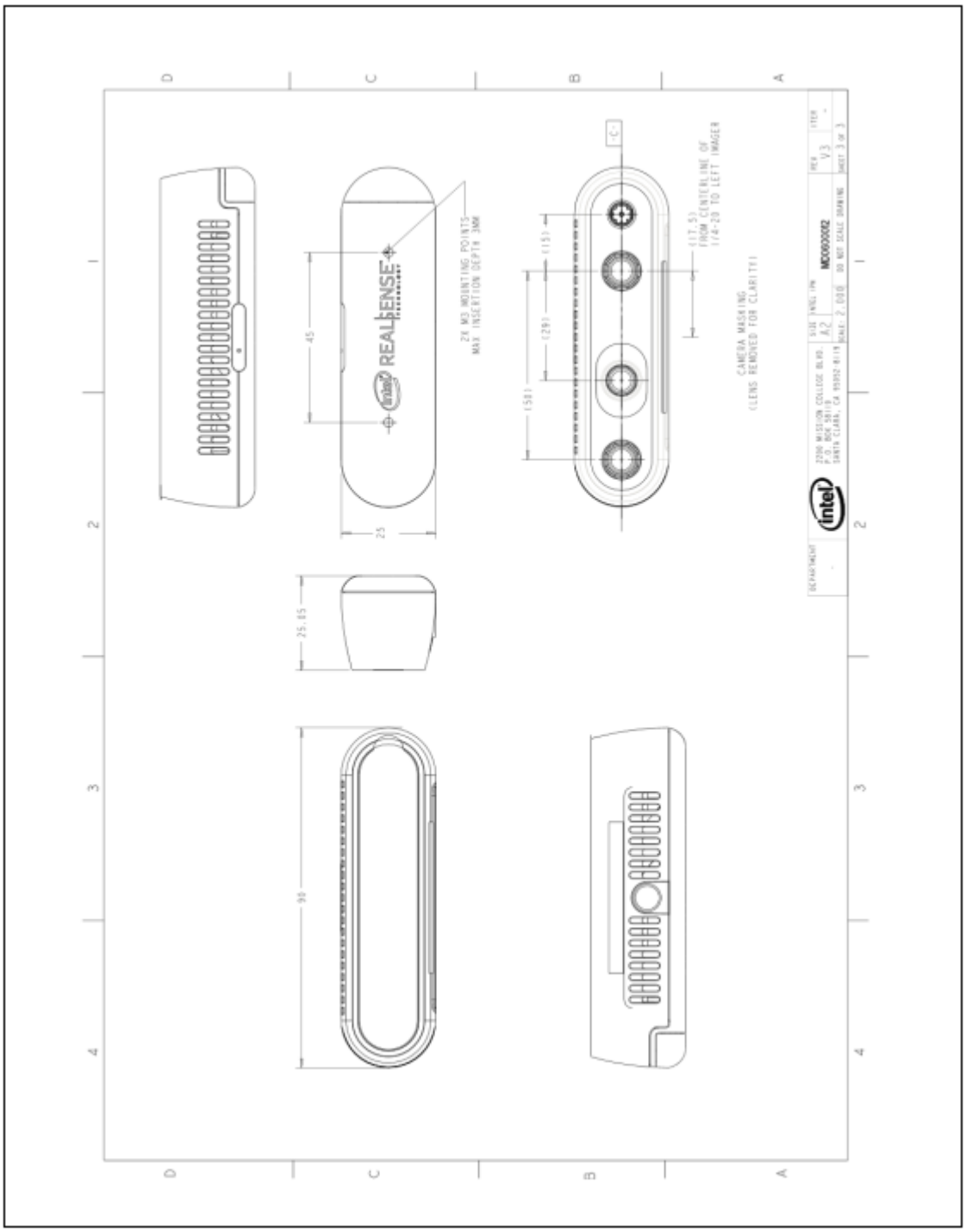
[147] A. Grunnet-jepsen, A. Takagi, J. Sweetser, T. Khuong, and D. Tong, “External Synchronization of Intel ® RealSense ™ Depth cameras 2 . Principles of Operation,” pp. 0–7.

[148] X. Gu, X. Wang, and Y. Guo, “A Review of Research on Point Cloud Registration Methods,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 782, no. 2, 2020, doi: 10.1088/1757-899X/782/2/022070.

[149] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognit.*, vol. 47, no. 6, pp. 2280–2292, 2014, doi: 10.1016/j.patcog.2014.01.005.

Appendix A Mechanical Drawings

Below is the mechanical drawing of the Intel RealSense D435 provided by Intel in their data sheet for the D400 series [93].



Appendix B MATLAB Code

B1 Data Acquisition

Data acquisition with the Intel RealSense D435 was performed using the following MATLAB functions and the Intel SDK 2.0 in the MATALB a C++ library wrapper. Specifically, the Intel RealSense D435 saved data to rosbag files that were read in MATLAB using the rs2ReadRosbag function. This function reads the rosbag file for data and converts it into a usable format. For example, the images are encoded as a vector that must be formatted and type cast into a numeric matrix. Also, the timestamps are stored as UTC 16-bit variables that are converted into date time using international standards. The rs2ReadRosbag function uses the SDK to post-process the depth maps. Primarily, the SDK is used to perform spatial and temporal filtering of the depth maps. Also, the SDK was used to align the color images and depth maps through projective transformation. All processed data is stored into a compressed .mat file for future use.

B1.1 rs2ReadRosbag

```
function[depthMap,colorImg,leftImg,rightImg,time,timestamp,depthIntrinsics,ma
tFilename,depthScale] =
rs2ReadRosbag(bagFilename,align,decParams,spatParams,tempParams,holeParams)
%% read .bag file from Intel Realsense Viewer recording for images and other
data
% function requires realsense+ library (MATLAB wrapper) for SDK 2.0
% countRosbagFrames() requires MATLAB ROS Toolbox
%
% EXAMPLE USEAGE
%
% % define absolute folder path to .bag file :
% bagFilename = 'D:\Masters\Research\Data\Mechanical Lung 2021-04-
09\Mechanical Lung ROS bag 2021-04-09\trial1.bag';
%
% % define optional inputs (spatParams and decParams use is recommended, and
is used by default):
% align = 'color';
% decParams = 2;
% spatParams = [0.5,20,2,0];
% tempParams = [0.4,20,3];
% holeParams = 1;
%
% [...] = rs2ReadRosbag(bagFilename); or any combination of optional inputs
% [...] =
rs2ReadRosbag(bagFilename,align,decParams,spatParams,tempParams,holeParams);
or
% [...] = rs2ReadRosbag(bagFilename,'',decParams,[],tempParams);
%
% INPUTS
```

```
%
% bagFilename = absolute folder path to .bag file with ASV recording
% align = optional input to return a synthetic image from a different ASV
persepctive
%
% % read for filter details : https://dev.intelrealsense.com/docs/post-
processing-filters
% decParams = optional input to use SDK 2.0 decimation filter on depth maps
% spatParams = optional input to use SDK 2.0 edge-preserving spatial
smoothing filter on depth maps
% tempParams = optional input to use SDK 2.0 temporal smoothing filter on
depth maps
% holeParams = optional input to use SDK 2.0 hole filling filter on depth
maps
%
% INTERMEDIATES
%
% the most ambiguous intermediates are the rs2 objects used for replaying the
recording
% read documentation for details :
https://intelrealsense.github.io/librealsense/doxygen/classrs2_1_1frame.html
% alternatively read example MATLAB scripts in realsense+ library :
https://dev.intelrealsense.com/docs/matlab-wrapper
% ie. cfg = SDK 2.0 configuration object that defines settings of device or
file interface
%
% OUTPUTS
%
% depthMap = stack of unscaled depth maps with dimensions [R1,C1,N] uint16
% colorImg = stack of RGB images with dimensions [R2,C2,3,N] uint8
% leftImg = stack of left ASV infrared images with dimensions [R1,C1,N] uint8
% rightImg = stack of right ASV infrared images with dimensions [R1,C1,N]
uint8
% time = backend time of when images were taken [N,1] double
% timestamp = datetime of when images were taken [N,1] datetime
% depthIntrinsics = struct of depth map ASV intrinsics for deprojection {1x1
struct}
% matFilename = folder path to compressed mat file with all results {string}
% depthScale = scaling for depth maps for : scaledDepthMap =
depthScale*double(depthMap);
%
% PROCESSING STEPS
%
% 1. create empty variables for images and data using the ROS Toolbox
% 2. replay .bag file using SDK 2.0
% 3. assign images and other data to memory
% 4. end replay once all frames have been read
% 5. remove dropped frames from all data types
% 6. save data to compressed mat file version 7.3
%
% NOTES
%
% bagFilename MUST BE ABSOLUTE FOLDER PATH does not work with relative
% updated version of rs2_rosbag_streams()
% alternative function is readRosbag() but it has memory issues and can not
use SDK filters
```

```
% known problem with rs2 replay feature is inconsistent frame dropping
% when using readRosbag notice that different ASVs have different number of
frames, but are the same when replaying
%
% Jason Der
% September 3, 2021 updated Feb 3, 2022 for efficiency and readibility
arguments
  bagFilename % needs full file path
  align char = 'notAligned' % 'color' or 'depth' or otherwise
  decParams = []
  spatParams = [0.5,20,2,0]
  tempParams = [0.4,20,3]
  holeParams = [] % enumerations
end
%% replay and read video
% initialize variables
[depthMap,leftImg,rightImg,colorImg,time,timestamp,nFrames] =
countRosbagFrames(bagFilename);
% check if frames are empty
depthEmpty = isempty(depthMap);
leftEmpty = isempty(leftImg);
rightEmpty = isempty(rightImg);
colorEmpty = isempty(colorImg);
% configure PIPe and replay
cfg = realsense.config();
Pipe = realsense.Pipeline();
cfg.enable_device_from_file(bagFilename,false);
profile = Pipe.start(cfg); % replay
depth_sensor = profile.get_device.first('depth_sensor');
playback = profile.get_device().as('playback');
playback.set_real_time(false);
% capture streams
frameDropped = false;
for iFrame = 1:nFrames
  try
    % if playback stopping
    if playback.current_status == 'stopped'
      fprintf('processed %.0i/%.0i frames\n',(iFrame-1),nFrames);
      frameDropped = true;
      break
    end
    % obtain streams from frameset
    [A,B,C,D,E,F,depthIntrinsics,colorIntrinsics,depthScale] = ...
      rs2PIPeStreams(PIPe,depth_sensor,align,...
      decParams,spatParams,tempParams,holeParams);
    % manage frameset output assignment
    if ~depthEmpty && ~isempty(A) depthMap(:,:,iFrame) = A; end
    if ~colorEmpty && ~isempty(B) colorImg(:,:,:,iFrame) = B; end
    if ~leftEmpty && ~isempty(C) leftImg(:,:,iFrame) = C; end
    if ~rightEmpty && ~isempty(D) rightImg(:,:,iFrame) = D; end
    % assign time and stamps without restriction
    time(iFrame) = E;
    timestamp(iFrame) = F;
  catch % error catch
    fprintf('processed %.0i/%.0i frames\n',(iFrame-1),nFrames);
    frameDropped = true;
```

```
    break
  end
  if iFrame == nFrames
    % exit loop, replay behaviour is not well understood
    % just a safety feature
    break
  end
end
% stop replay
PIPe.stop();
%% remove dropped frames
if frameDropped % if frame[s] dropped
% check which frames are blank images
depthBlank = ~squeeze(any(depthMap,[1 2]));
colorBlank = ~squeeze(any(colorImg,[1 2 3]));
leftBlank = ~squeeze(any(leftImg,[1 2]));
rightBlank = ~squeeze(any(rightImg,[1 2]));
timeBlank = ~any(time,2);
% common dropped frames
droppedFrames = depthBlank & colorBlank & leftBlank & rightBlank & timeBlank;
% assign empty to frames that are blank
depthMap(:,:,droppedFrames) = [];
colorImg(:,:,:,droppedFrames) = [];
leftImg(:,:,droppedFrames) = [];
rightImg(:,:,droppedFrames) = [];
time(droppedFrames) = [];
timestamp(droppedFrames) = [];
end
%% save to .mat file
% offset to zero
t = (time - time(1))/1000;
% convert datetime into numeric, then offset
T = datenum(timestamp)-datenum(timestamp(1));
% generate mat file name
[path,name] = fileparts(bagFilename);

if strcmp(align,'color') || strcmp(align,'depth')
  streamName = [name 'Aligned' align '.mat'];
else
  streamName = [name '.mat'];
end
matFilename = fullfile(path,streamName);
% save
save(matFilename,'matFilename','colorImg','leftImg','rightImg','depthMap',...

'time','timestamp','t','T','depthIntrinsics','colorIntrinsics','depthScale','
-v7.3');
End
```

B1.2 countRosbagFrames

```
function [depthMap,leftImg,rightImg,colorImg,time,timestamp,nFrames] =
countRosbagFrames(filename)
```

```
%% read ROS bag file for number for frames
% requires ROS bag Toolbox
% was called frameCount_Rosbag changed Feb 18, 2022
% Jason Der
% April 7, 2021 updated Febuary 3, 2022 to read frame size too and
% initialzie frameset
% based on: another personal custom function [readRosbag]
  % source: https://community.intel.com/t5/Items-with-no-label/Converting-
bag-video-to-raw-frames/m-p/509808
  % alternative : https://github.com/UnaNancyOwen/rs_bag2image
% based on:
https://github.com/IntelRealSense/librealsense/issues/6887#issuecomment-
662310354
% Displays information in the command window. Use bagInfo = rosbag('info',
'file.bag') to get the information as a structure in a script. See
https://github.com/IntelRealSense/librealsense/blob/master/src/media/readme.m
d%23under-the-hood
https://github.com/IntelRealSense/librealsense/blob/master/src/media/readme.m
d# under-the-hood for explanations.
% Select a specific topic from the above information. It should end with
/image/data if you want the frames.
%% read frame data topics
arguments
  filename char % absolute path to .bag file
end
% Load the rosbag into object
bag = rosbag(filename);
% read topics
leftTopic = select(bag, 'Topic', '/device_0/sensor_0/Infrared_1/image/data');
rightTopic = select(bag, 'Topic',
'/device_0/sensor_0/Infrared_2/image/data');
colorTopic = select(bag, 'Topic', '/device_0/sensor_1/Color_0/image/data');
depthTopic = select(bag, 'Topic', '/device_0/sensor_0/Depth_0/image/data');
% read number of msgs as number of frames
leftFrameCount = leftTopic.NumMessages;
rightFrameCount = rightTopic.NumMessages;
colorFrameCount = colorTopic.NumMessages;
depthFrameCount = depthTopic.NumMessages;
% read image msgs for image size
if depthFrameCount ~= 0
  depthMsgs = readMessages(depthTopic,1);
  depthFrameSize = [depthMsgs{1}.Height,depthMsgs{1}.Width];
else depthFrameSize = [0 0]; end
if colorFrameCount ~= 0
  colorMsgs = readMessages(colorTopic,1);
  colorFrameSize = [colorMsgs{1}.Height,colorMsgs{1}.Width];
else colorFrameSize = [0 0]; end
if leftFrameCount ~= 0
  leftMsgs = readMessages(leftTopic,1);
  leftFrameSize = [leftMsgs{1}.Height,leftMsgs{1}.Width];
else leftFrameSize = [0 0]; end
if rightFrameCount ~= 0
  rightMsgs = readMessages(rightTopic,1);
  rightFrameSize = [rightMsgs{1}.Height,rightMsgs{1}.Width];
else rightFrameSize = [0 0]; end
clear leftTopic rightTopic colortopic depthTopic bag
```

```
% determine number of frames to initialize
nFrames =
max([depthFrameCount,colorFrameCount,leftFrameCount,rightFrameCount]);
if nFrames > 16000
  % limit based on 15.9 GB allocation limit per variable
  % depthMap will hit this limit first around 16000 frames using uint16 at
848 x 480
  disp(fprintf('nFrames %.0i would cause allocation error, set to
4000\n',nFrames));
  nFrames = 16000;
end
% initialize images
depthMap = zeros([depthFrameSize,nFrames],'uint16');
leftImg = zeros([leftFrameSize,nFrames],'uint8');
rightImg = zeros([rightFrameSize,nFrames],'uint8');
colorImg = zeros([colorFrameSize,3,nFrames],'uint8');
time =
datetime(zeros(nFrames,1),0,0,'TimeZone','America/Denver','Format','yyyy-MM-
dd HH:mm:ss.SSS');
timestamp =
datetime(zeros(nFrames,1),0,0,'TimeZone','America/Denver','Format','yyyy-MM-
dd HH:mm:ss.SSS');
end
```

B1.2.1 rs2pipeStreams

```
function[depthMap,colorImg,leftImg,rightImg,time,timestamp,depthIntrinsics,co
lorIntrinsics,depthScale] =
rs2PIPeStreams(PIPe,depth_sensor,align,decParams,spatParams,tempParams,holePa
rams)
%% read pipe object for image and property data
% Jason Der
% September 3, 2021
% NOTE:
  % better version of rs2_streams()
  % requires realsense SDK 2.0
arguments
  pipe
  depth_sensor
  align = 'color'
  decParams = []
  spatParams = []
  tempParams = []
  holeParams = []
end
% get frameset
fs = PIPe.wait_for_frames();
% read and process frameset
alignedFs = rs2AlignFrameset(fs,align);
[depth,color,left,right,time,timestamp] = rs2ReadFrameset(alignedFs);
filtered = rs2FilterDepth(depth,decParams,spatParams,tempParams,holeParams);
% read depth frame
try depthIntrinsics = rs2DepthIntrinsics(filtered); depthScale =
depth_sensor.get_depth_scale();
depthMap = rs2FrameImg(filtered,'depth'); % depthMap = depthScale .*
depthMap;
catch depthIntrinsics = []; depthMap = []; end
```

```
% read color frame
try colorIntrinsics = rs2DepthIntrinsics(color); colorImg =
rs2FrameImg(color,'color');
catch colorIntrinsics = []; colorImg = []; end
% read infrared frame
try leftImg = rs2FrameImg(left,'infrared'); catch leftImg = []; end
try rightImg = rs2FrameImg(right,'infrared'); catch rightImg = []; end
end
```

B1.1.1.1 rs2AlignFrameset

```
function [aligned] = rs2AlignFrameset(fs,align)
%% synthetically align frameset to color or depth persepctive
% Jason Der
% September 3, 2021
% NOTE:
  % align == 'color' or 'depth', otherwise no alignment
  % requires intelRealSense SDK 2.0
arguments
  fs
  align
end
% align frameset to selected ASV
switch align
  case 'color'
    colorAlign = realsense.align(realsense.stream.color);
    aligned = colorAlign.process(fs);
  case 'depth'
    depthAlign = realsense.align(realsense.stream.depth);
    aligned = depthAlign.process(fs);
  otherwise
    aligned = fs;
end
end
```

B2.1.1.1 rs2ReadFrameset

```
function [depth,color,left,right,time,timestamp] = rs2ReadFrameset(fs)
%% read rs2 PIPe object for enabled streams frame objects, frame time, and
frame timestamp
% back end timestamp is taken from depth
% Jason Der
% September 3, 2021
% NOTE:
  % requires intelRealSense SDK 2.0
  %
https://intelrealsense.github.io/librealsense/doxygen/classrs2_1_1frameset.ht
ml
  %
https://intelrealsense.github.io/librealsense/doxygen/rs__frame_8h.html#a91f1
9a01f5bf2abadc30959a8d3109c9
% READ
  % frame.get_timestamp - explains timestamps
  % frame.get_frame_metadata(metadatavalue) - other timestamps are available
in meta data
  % meta data value chart - list of meta data including timestamps
% frame.get_timestamp() will dynamically choose most appropriate timestamp
  % it will choose between timestamps at device and host level
```

```
  % use get_frame_timestamp_domain to identify which it selected
% timestamps available under meta data
  % metadata_frame_timestamp
  % metadata_sensor_timestamp
  % metadata_time_of_arrival
  % metadata_backened_timestamp
% timestamp domains and enumerators
  % 1 - hardware clock
  % 2 - system time
  % 3 - global time
  % also - domain count - number of enuermation values (NOT A VALID INPUT)

arguments
  fs
end

%% get frameset and time properties
left = fs.get_infrared_frame(1);
right = fs.get_infrared_frame(2);
color = fs.get_color_frame();
depth = fs.get_depth_frame();

%% get time properties
% timestamps are milliseconds
% automatically selected appropriate timestamp
time = fs.get_timestamp();
time = datetime(time,'TimeZone','UTC',...
      'ConvertFrom','epochtime','TicksPerSecond',1000, 'Format', 'yyyy-MM-dd
HH:mm:ss.SSS');
time.TimeZone = 'America/Denver';
% backend timestamp
try
  backend_timestamp =
depth.get_frame_metadata(realsense.frame_metadata_value.backend_timestamp);
catch
  backend_timestamp =
left.get_frame_metadata(realsense.frame_metadata_value.backend_timestamp);
end
timestamp = datetime(backend_timestamp,'TimeZone','UTC',...
      'ConvertFrom','epochtime','TicksPerSecond',1000, 'Format', 'yyyy-MM-dd
HH:mm:ss.SSS');
timestamp.TimeZone = 'America/Denver';

end
```

B1.2.2 rs2FilterDepth

```
function [filtered] =
rs2FilterDepth(depth,decParams,spatParams,tempParams,holeParams)
%% post process depth frame object
% Jason Der
% September 3, 2021
% NOTE:
  % to exclude filter set argument setting = []
  % requires intelRealSense SDK 2.0
```

```
  arguments
    depth
    decParams = []
    spatParams = []
    tempParams = []
    holeParams = []
  end

%% post process depth frame object with filter blocks
  [decimation,spatial,temporal,hole,depth_to_disparity,disparity_to_depth] =
rs2_filters(decParams,spatParams,tempParams,holeParams);
  filtered = depth;
  % decimation filter
  if ~isempty(decParams)
    filtered = decimation.process(filtered);
  end
  % spatial and temporal filter in disparity domain
  if (~isempty(spatParams) || ~isempty(tempParams)) &&...
     (~isempty(depth_to_disparity) && ~isempty(disparity_to_depth))
    filtered = depth_to_disparity.process(filtered);
    if ~isempty(spatParams)
      filtered = spatial.process(filtered);
    end
    if ~isempty(tempParams)
      filtered = temporal.process(filtered);
    end
    filtered = disparity_to_depth.process(filtered);
  end
  % hole filling filter
  if ~isempty(holeParams)
    filtered = hole.process(filtered);
  end
  % ensure object is realsense.depth_frame
  if ~strcmp(class(filtered),'realsense.depth_frame')
    filtered = filtered.as('depth_frame');
  end

end
```

B1.2.2.1 rs2_filters

```
function
[decimation,spatial,temporal,hole,depth_to_disparity,disparity_to_depth] =
rs2_filters(dec_mag,spat_settings,temp_settings,hole_settings)
%% create and configure
% Jason Der
% June 10, 2021
% NOTE: to exclude a filter set settings argument = []

%% function argument validation
  arguments
    dec_mag {mustBeReal,mustBeFinite} = 2
    spat_settings {mustBeReal,mustBeFinite} = [0.5,20,2,0]
    temp_settings {mustBeReal,mustBeFinite} = [0.4,20,3]
    hole_settings {mustBeReal,mustBeFinite} = 0
  end
```

```
%% initialization of post-processing filters for depthmap
  decimation = realsense.decimation_filter;
  spatial = realsense.spatial_filter;
  temporal = realsense.temporal_filter;
  hole = realsense.hole_filling_filter;
  depth_to_disparity = realsense.disparity_transform(true);
  disparity_to_depth = realsense.disparity_transform(false);

%% configure filters
  % if decimation filter exists
  if ~isempty(decimation) && ~isempty(dec_mag)
    % configure decimation filter linear scale factor
    decimation.set_option(realsense.option.filter_magnitude,dec_mag);
  end

  % if spatial filter exists
  if ~isempty(spatial) && ~isempty(spat_settings)
    % configure spatial filter factor for exponential moving average

spatial.set_option(realsense.option.filter_smooth_alpha,spat_settings(1));
    % configure spatial filter step size boundary, threshold to preserve
edges

spatial.set_option(realsense.option.filter_smooth_delta,spat_settings(2));
    % configure spatial filter filter iterations
    spatial.set_option(realsense.option.filter_magnitude,spat_settings(3));
    % configure spatial filter rectify minor artefacts
    spatial.set_option(realsense.option.holes_fill,spat_settings(4));
  end

  % if temporal filter exists
  if ~isempty(temporal) && ~isempty(temp_settings)
    % configure temporal filter factor for exponential moving average

temporal.set_option(realsense.option.filter_smooth_alpha,temp_settings(1));
    % configure temporal filter step size boundary, threshold to preserve
edges

temporal.set_option(realsense.option.filter_smooth_delta,temp_settings(2));
    % configure temporal filter governs hole filling based on historic pixel
value
    temporal.set_option(realsense.option.holes_fill,temp_settings(3));
  end

  % if hole filling filter exists
  if ~isempty(hole) && ~isempty(hole_settings)
    % configure hole filling filter filling type
    hole.set_option(realsense.option.holes_fill,hole_settings);
  end

end
```

B1.2.3 rs2DepthIntrinsics

```
function [intrinsics] = rs2DepthIntrinsics(depth)
```

```
%% get depth frame object intrinsics
% Jason Der
% September 3, 2021

  arguments
    depth
  end

%% get intrinsics property
  profile = depth.get_profile();
  video_profile = profile.as('video_stream_profile');
  intrinsics = video_profile.get_intrinsics();

end
```

B1.2.4 rs2FrameImg

```
function [img] = rs2FrameImg(frame,type)
%% create image from frame object
% Jason Der
% September 3, 2021
% NOTE:
  % requires intelRealSense SDK 2.0
  % if depth, need to multiply img by depthScale
  %
https://intelrealsense.github.io/librealsense/doxygen/classrs2_1_1frame.html
  % realsense.frame does not have get_width, get_height methods
  % these methods are inherited from realsense.video_frame
  % realsense.video_frame is used for infrared and color
  % realsense.depth_frame is used for depth
  % realsense.frame is used for filtered realsense.depth_frame

  arguments
    frame
    type = []
  end

%% create image, format based on data size
  try
    % ensure object is realsense.video_frame or realsense.depth_frame
    if strcmp(class(frame),'realsense.depth_frame') ||
strcmp(class(frame),'realsense.video_frame')
      try
        frame = frame.as('video_frame');
      catch
        frame = frame.as('depth_frame');
      end
    end
    % identify missing format type
    if isempty(type)
    profile = frame.get_profile;
    streamType = profile.stream_type;
      switch streamType
        case 1
          type = 'depth';
        case 2
```

```
          type = 'color';
        case 3
          type = 'infrared';
      end
    end
    % format frame data into image
    switch type
      case 'infrared' % uint8
        img =
permute(reshape(frame.get_data()',[frame.get_width(),frame.get_height()]),[2
1]);
      case 'color' % uint8
        img =
permute(reshape(frame.get_data()',[3,frame.get_width(),frame.get_height()]),[
3 2 1]);
      case 'depth' % save as uint16, float(uint16) x depthScale for actual
use
        img = (transpose(reshape(frame.get_data(),
[frame.get_width(),frame.get_height()])));
    end
  % entire process try-catch
  catch
    img = [];
  end
end
```

B2 Image Segmentation

The various experiment lungs were segmented from the depth maps using an edge-based or region-based segmentation method. The segmentMechanicalLung function performed edge-based segmentation of the ventilator test lung depth maps. The segmentLungs function performed the interactive region-based segmentation of the porcine lung and rejected human lung using the projective transformed color images. Also, the left and right lung segmentation of the human lung was performed using the segmentRegions function. Segmentation allowed the isolation of the experiment lungs in the depth maps for only their depth measurements. Also, the label and color maps were segmented using the segmentation binary maps.

B2.1 segmentLungs

```
function [segmentationMatFilename] =
segmentLungs(streamsMatFilename,depthThreshold)
%% segment color image for lung, then remove invalid data from segmented
depthMap
% Jason Der
% October 29, 2021
arguments
  streamsMatFilename
  depthThreshold = [] % set based on foreground and background
end
```

```
% color image segmentation
load(streamsMatFilename,'-mat','colorImg');
[colorMask,forePosition,backPosition] = roiSegmentationVectorized(colorImg);
objectMask = largestRegion(colorMask);
refinedMask = refineMask(objectMask);
temporalMask = temporalFilterMask(refinedMask);
clear colorImg
% depthMap segmentation
load(streamsMatFilename,'-mat','depthMap');
if isempty(depthThreshold)
  depthThreshold = graythresh(depthMap(:,:,1));
end
depthMask = segmentDepth(depthMap,temporalMask,depthThreshold);
clear depthMap
% save and clear variables
[file,name,ext] = fileparts(streamsMatFilename);
segmentationMatFilename = fullfile(file,[name 'Segmentation' ext]);
%
save(segmentationMatFilename,'colorMask','objectMask','refinedMask','temporal
Mask','depthMask');
save(segmentationMatFilename,'colorMask','objectMask','refinedMask',...
  'temporalMask','depthMask','forePosition','backPosition','depthThreshold');
end
```

B2.1.1 roiSegmentationVectorized

```
function [mask,forePosition,backPosition] =
roiSegmentationVectorized(img,foreground,background,type,labelSuperPixels,com
pactness,nIterations)
%% segment color image
% Jason Der
% September 14, 2021
% NOTES:
  % consider using 3D volumetric image processing:
    % https://www.mathworks.com/help/images/3d-volumetric-image-
processing.html?s_tid=CRUX_lftnav
  % consider using sparse for masks, even though logical data is small
  % use imfilter and fspecial for enhancement

  arguments
    img
    foreground = []
    background = []
    type = 'color' % or depth
    labelSuperPixels (1,1) {mustBePositive} = 500
    compactness (1,1) {mustBePositive} = 10
    nIterations (1,1) {mustBePositive} = 10
  end

%% interactively define foreground and background
  switch type
    case 'depth'
      previewImg = img(:,:,1);
    case 'color'
      previewImg = img(:,:,:,1);
  end
  if isempty(foreground) || isempty(background)
```

```
    % open first image in figure
    f = figure;
    set(gcf,'color','w');
    imshow(rescale(previewImg,0,1));
    title('first color image - select ROI with only foreground');
    foreROI = drawrectangle(gca);
    forePosition = foreROI.Position;
    foreground = createMask(foreROI);
    title('first color image - select ROI with only background');
    backROI = drawrectangle(gca);
    backPosition = backROI.Position;
    background = createMask(backROI);
    close(f);
  end

%% enhance images
  img = imadjustn(img);
  filter = fspecial('unsharp');
  img = imfilter(img,filter);

%% segment images
  switch type
    case 'depth'
      mask = false(size(img));
      for i = 1:size(img,3)
        [labelImg,~] =
superpixels(img(:,:,i),labelSuperPixels,'Compactness',compactness,'NumIterati
ons',nIterations);
        mask(:,:,i) =
lazysnapping(img(:,:,i),labelImg,foreground,background);
      end
    case 'color'
      mask = false(size(img,1,2,4));
      for j = 1:size(img,4)
        [labelImg,~] =
superpixels(img(:,:,:,j),labelSuperPixels,'Compactness',compactness,'NumItera
tions',nIterations);
        mask(:,:,j) =
lazysnapping(img(:,:,:,j),labelImg,foreground,background);
      end
  end

end
```

B2.1.2 largestRegion

```
function [bw] = largestRegion(BW)
bw = BW;
for ii = 1:size(BW,3)
  bw(:,:,ii) = bwpropfilt(BW(:,:,ii),'Area',1);
end
```

B2.1.3 refineMask

```
function [maskOut] = refineMask(maskIn)
%% refine binary segmentation mask
% Jason Der
```

```
% September 9, 2021, updated September 14, 2021 for ND masks
  arguments
    maskIn
  end
%% initializations
  s = strel('disk',3,8);
  n = size(maskIn,3);
  maskOut = false(size(maskIn));
%% refine mask
  for i = 1:n
%     % remove regions that contact image border
%     mask = imclearborder(maskIn(:,:,i));
    % morphological operations
    mask = bwmorph(maskIn(:,:,i),'fill');
    mask = bwmorph(mask,'clean');
    mask = imerode(mask,s);
    mask = bwmorph(mask,'thin');
    mask = bwmorph(mask,'hbreak');
    mask = bwmorph(mask,'spur');
    mask = bwmorph(mask,'clean');
    maskOut(:,:,i) = imdilate(mask,s);
  end
end
```

B2.1.4 temporalFilterMask

```
function [maskOut] = temporalFilterMask(maskIn,window,threshold)
%% use temporal moving average mask
% takes any type of image, but outputs a logical mask
% Jason Der
% September 22, 2021
% other options are imfilter, or filter, and fspecial
  % fspecial could use 'motion', 'average', 'gaussian'

arguments
  maskIn
  window (1,1) {mustBePositive} = floor(0.01*size(maskIn,3))
  threshold (1,1) {mustBeNonnegative} = 0.5 % larger, more sensitive to
change
end

%% filter through frames
[~,~,i,j] = size(maskIn);
if i == 3 && j ~=1 && all(class(maskIn) == 'uint8') % color image
  n = 4;
else % grayscale or binary
  n = 3;
end
maskOut = movmean(maskIn,window,n,'omitnan') > threshold;
end
```

B2.1.5 segmentDepth

```
function [depthMask] = segmentDepth(depth,colorMask,threshold,direction)
%% segment depthMap for depthMap where AND(color image mask, above/below
threshold, continuous object)
% Jason Der
% October 31, 2021
```

```
arguments
  depth double
  colorMask logical
  threshold = abs(mean(depth.*colorMask,'all') +
2*std(depth.*colorMask,0,'all'))
  direction logical = true % default find depth less than threshold
end
% segment depthMap, and find continuous objects
depthSeg = depth.*colorMask;
object = largestRegion(depthSeg & colorMask);
% find depth above/below threshold
if direction % depth greater than threshold
  maskThreshold = ~(depthSeg > threshold);
else % depth less than threshold
  maskThreshold = depthSeg < threshold;
end
% binary mask meeting all three conditions
depthMask = depthSeg & maskThreshold & object;
% return largest contiguous regions
depthMask = largestRegion(depthMask);
end
```

B2.1.6 segmentMechanicalLungEVLP

```
function [bwFilename] = segmentMechanicalLungEVLP(matFilename,line,rect)
%% segment mechanical lung from depth maps
% uses edge based method and edge linking from Digital Image Processing 4th
Ed.
% edge localization by bwG(:,:,ii) =
gradientEdge(depthMap(:,:,ii),0.95,60,90)
% or edge(depthMap(:,:,ii),'canny') or edge(depthMap(:,:,ii),'log')
% other methods covered by edge() could be used but these are recommended

% method is finicky
% depends on small body filtering threshold
  % only used on trials 4-6 limits = [15 inf]
  % trials 1-3 dont need it
  % trials 7-9 need the small bodies
% depends on location of drawline and drawrect
  % trials 1-3 vertical line good
  % trials 4-6 needed close diagonal line or else it would not enclose lung
% depends on linking distance
  % trials 1-3 threshold = 10 was good
  % trials 4-6 threshold = 10 with area filt prior okay
  % trials 7-9 threshold = 10 lost > 5% of frames

% Jason Der
% February 17, 2022
arguments
  matFilename
  line = []
  rect = []
end
% load depthmaps
load(matFilename,'depthMap');
[file,name,ext] = fileparts(matFilename);
bwFilename = fullfile(file,[name 'BW' ext]);
```

```
[row,col,nFrames] = size(depthMap);
% edge localization and linking
bwCanny = false(row,col,nFrames);
parfor iFrame = 1:nFrames
  bwCanny(:,:,iFrame) = edge(imadjust(depthMap(:,:,iFrame)),'canny');
end
% remove small edges
% for trial 4 aimed at occlusion holes scattered throughout image
% for trial 4 lower limit ~ 15, above will remove too many that are needed
% for enclose the lung, at ~ 10 it has no effect
% bwAreaFilt = false(row,col,nFrames);
% for i = 1:nFrames
%   bwAreaFilt(:,:,i) = bwareafilt(bwCanny(:,:,i),[15 inf]);
% end
save(bwFilename,'bwCanny','-v7.3');
clear file name ext matFilename
bwLinked = false(row,col,nFrames);
parfor jFrame = 1:nFrames
  bwLinked(:,:,jFrame) = fastLocalEdgeLinking(bwCanny(:,:,jFrame),10,[0 45 -
45 90]);
end
% save
save(bwFilename,'bwLinked','-append'); % change based on edge detector
clear bwAreaFilt .
% define boundaries of mechanical lung airline and crop
if isempty(line) || isempty(rect)
  [line,rect] = drawMechanicalLungBounds(bwLinked(:,:,1));
end
% find mechanical lung within cropped image
bwCrop = false(rect(4)+1,rect(3)+1,nFrames);
for kFrame = 1:nFrames
  bwCrop(:,:,kFrame) =
mechanicalLungRegionInsideBounds(bwLinked(:,:,kFrame),line,rect);
end
clear bwLinked
% outlier detection
bwCount = squeeze(sum(bwCrop,[1 2]));
bwOutlier = isoutlier(bwCount,'movmedian',0.01*length(bwCount));
bwValid = bwCrop(:,:,~bwOutlier);
save(bwFilename,'bwCrop','bwOutlier','rect','line','-append');
clear bwCrop bwCount bwOutlier line
% bw temporal filtering on cropped binary images
bwCropFilt = temporalFilterMask(bwValid,10,0.5); % or temporalFilterImg
% check if outlier removal and temporal smoothing worked
% bwFiltCount = squeeze(sum(bwCropFilt,[1 2]));
% bwFiltOutlier =
isoutlier(bwFiltCount,'movmedian',0.01*length(bwFiltCount));
% insert cropped binary images into full sized images
mValidFrames = size(bwCropFilt,3);
bw = false(row,col,mValidFrames);
for kFrame = 1:mValidFrames
  bw(rect(2):rect(4)+rect(2),rect(1):rect(3)+rect(1),kFrame) =
bwCropFilt(:,:,kFrame);
end
% save
save(bwFilename,'bw','bwCropFilt','-append');
```

```
end
```

B2.1.6.1 fastLocalEdgeLinking

```
function [bwE] = fastLocalEdgeLinking(bwG,L,theta)
%% perform local processing edge linking
% based on simplified algorithm described in Digital Image Processing 4th Ed.
pg. 736
% find bwG using any edge detector such as gradientEdge()
% inputs
  % bwG binary image from edge detection
  % L edge linking distance threshold
  % theta desired edge angles to link
% intermediates
  % n number of nonzero pixels in row(ii) of bwG
  % ind linear indices of nonzero pixels in row(ii) of bwG
  % dist number of pixels between nonzero pixels in row(ii) of bwG
% outputs
  % bwE binary image with linked edges
% Jason Der
% February 16, 2022
arguments
  bwG
  L = 15
  theta = [0 45 -45 90]
end
% pre-edge linking using morphological operations
bwG = bwmorph(bwG,'thicken');
bwG = bwmorph(bwG,'diag');
bwG = bwmorph(bwG,'majority');
% fill gaps shorter than L for each row
bwE = bwG;
for kk = 1:length(theta)
  % rotate so theta(kk) angle is horizontal
  bwE = imrotate(bwE,theta(kk),'crop');
  bwG = imrotate(bwG,theta(kk),'crop');
  for ii = 1:height(bwG) % rows
    % find nonzero pairs and their distances
    n = nnz(bwG(ii,:));
    ind = find(bwG(ii,:));
    dist = diff(ind);
    for jj = 1:n-1 % nonzero pairs
      if dist(jj) <= L % threshold pair distance
        % fill gaps between nonzero pair
        bwE(ii,ind(jj):ind(jj+1)) = true;
      end
    end
  end
  % rotate back
  bwE = imrotate(bwE,-theta(kk),'crop');
  bwG = imrotate(bwG,-theta(kk),'crop');
end
% edge thinning
bwE = bwmorph(bwE,'diag');
bwE = bwmorph(bwE,'skel',inf);
% remove small bodies
bwE = bwareafilt(bwE,[50 inf]);
```

```
end
```

B2.1.7 segmentRegions

```
function [segmentationMatFilename] =
segmentRegions(streamsMatFilename,segmentationMatFilename)
%% segment lung for regions, left and right lungs
% Jason Der
% October 29, 2021
% updated November 17, 2021 with clusterLabels
arguments
  streamsMatFilename
  segmentationMatFilename
end
% load variables
load(segmentationMatFilename,'depthMask');
load(streamsMatFilename,'depthMap');
% segment regions
[r,c,n] = size(depthMap);
L = ones(r,c,n,'uint8');
parfor i = 1:n
  [L(:,:,i)] = segmentRegionsImquantize(depthMap(:,:,i),8,depthMask(:,:,i));
end
clear depthMap depthMask
% post process regions
Lc = clusterLabels(L,3);
Lw = removeLabelIdx(Lc);
Ls = sortLabels(Lw);
% temporal filter region
% Lf = filterRegion(Ls);
Lf = filterLeftRightLungLabels(Ls,[2 3 1],0.03);
% save to mat file
save(segmentationMatFilename,'L','Lc','Lw','Ls','Lf','-append');
end
```

B2.1.7.1 segmentRegionsImquantize

```
function [L] = segmentRegionsImquantize(img,nLevels,mask)
%% segment regions from result of imquantize to seed watershed method
% Jason Der
% November 2, 2021
% updated November 17, 2021 with multithresh, default nLevels = 9 from 10
arguments
  img
  nLevels = 8
  mask = []
end
% prepare image for quantization
if ~isempty(mask)
  seg = img.*imfill(mask,'holes');
else
  seg = img;
end
nonZero = imfill(seg);
nonZero(nonZero==0) = max(seg,[],'all');
% quantize image into nLevels
thresh = multithresh(nonZero,nLevels);
% levels = linspace(min(nonZero,[],'all'),max(nonZero,[],'all'),nLevels);
```

```
quant = imquantize(nonZero,thresh);
% find watershed seed regions, foreground and background
minima = imhmin(quant,2);
foreground = bwmorph(imregionalmin(minima),'clean');
foreground = imerode(foreground,strel('disk',3));
  if ~isempty(mask)
    background = ~imfill(mask,'holes');
  else
  maxima = imhmax(quant,1);
  background = bwmorph(imregionalmax(maxima),'clean');
  end
%gradient image with imposed minima
smooth = imnlmfilt(imsharpen(nonZero));
gmag = imgradient(smooth);
gmag2 = imimposemin(gmag,foreground|background);
% watershed segmentation
L = watershed(gmag2);
end
```

B2.1.7.2 clusterLabels

```
function [clusterLabel] = clusterLabels(L,nClusters,attribute)
%% ensure the same regions have the same labels between frames
% also combines oversegmented regions
% identify unique regions by clustering based on a metric
% metrics could be any supported regionprops attributes (choosen is Area)
% attribute input is case sensitive
% k-means function is set to use replicates = 3
% Jason Der
% November 16, 2021 updated March 4, 2022 to use Area instead of Centroid
arguments
  L
  nClusters = 3
  attribute = 'Area'
end
% initialize variables
[r,c,n] = size(L);
s = cell(n,1);
frameID = cell(n,1);
labelID = cell(n,1);
clusterLabel = zeros(r,c,n,'uint8');
% regionprops for each frame
for ii = 1:n
  % regionprops ignores zero label
  % zero label is watershed line (between regions)
  s{ii} = regionprops(L(:,:,ii),attribute);
  frameID{ii} = repmat(ii,length(s{ii}),1);
  labelID{ii} = uint8(repmat(1:length(s{ii}),1))';
end
% measure dimensions of attribute
attributeLength = length(s{1}(1).(attribute));
% create matching arrays to clusterID that tells you the frame and label
frameID = cell2mat(frameID);
labelID = cell2mat(labelID);
% identify number of k-mean clusters
if isempty(nClusters)
  labels = uint8(unique(L));
```

```
  labels(labels==0) = []; % remove watershed lines
  nClusters = length(labels);
end
% identify cluster for each region for each frame
regionProperties =
reshape(cell2mat(struct2cell(cell2mat[s])),attributeLength,[])';
clusterID = uint8(kmeans(regionProperties,nClusters,'Replicates',3));
% create label matrix for cluster ID regions
for jj = 1:length(clusterID)
  frame = frameID(jj);
  label = labelID(jj);
  cluster = clusterID(jj);
  clusterLabel(:,:,frame) = clusterLabel(:,:,frame) +
cluster.*uint8(L(:,:,frame)==label);
end
end
```

B2.1.7.3 removeLabelIdx

```
function [Lw] = removeLabelIdx(Lc,filtSz,watershedLabel,backgroundLabel)
%% remove watershed lines between regions
% Jason Der
% December 9, 2021
arguments
  Lc
  filtSz = 9
  watershedLabel = 0
  backgroundLabel = 3
end
% apply majority filter to label image
majority = modefilt(Lc,[filtSz filtSz 1]);
% use majority filter results for pixels are equal to watershedIdx
Lw = Lc;
loc = Lc == watershedLabel;
Lw(loc) = majority(loc);
% remove any remaining watershed pixels
locw = (Lw == watershedLabel);
if any(locw,'all')
Lw(locw) = backgroundLabel; % take as background label
end
```

B2.1.7.4 sortLabels

```
function [Ls] = sortLabels(L)
%% change region labels by sorting them by pixel count / frequency
% intended to avoid pcdownsampling a background label
% utility assumes that background has larger count than foreground
% as a result, the background label is always known and interpolation
% between foreground labels when rounded always gives a foreground label
% Jason Der
% March 5, 2022
arguments
  L
end
% sort labels in first labelling image
firstFrame = L(:,:,1);
[groupCount,groupLabels] = groupcounts(firstFrame(:));
[~,I] = sort(groupCount);
```

```
sortedLabels = groupLabels(I);
% sort all labelling images based on sorted groups
Ls = zeros(size(L),'uint8');
for iGroup = 1:length(groupLabels)
  Ls(L==sortedLabels(iGroup)) = groupLabels(iGroup);
end
end
```

B2.1.7.5 filterLeftRightLungLabels

```
function [Lf] = filterLeftRightLungLabels(Ls,labels,windowPercent)
%% temporal filter left and right lungs in labelling image
% Jason Der
% March 14, 2022
arguments
  Ls
  labels = [1 2 3]
  windowPercent = 0.03
end
% initialization
[r,c,n] = size(Ls);
window = floor(windowPercent*n);
leftLabel = labels(1);
rightLabel = labels(2);
backLabel = labels(3);
% moving mean of left and right lung
leftMean = movmean(Ls==leftLabel,window,3,'omitnan');
rightMean = movmean(Ls==rightLabel,window,3,'omitnan');
% assign foreground as either left or right lung
Lf = zeros(r,c,n,'uint8');
Lf(leftMean<=rightMean) = rightLabel;
Lf(leftMean>rightMean) = leftLabel;
% assign background
Lf(Ls==backLabel) = backLabel;
% find isolated bodies and replace for left lung
[ccIsolatedLeft,~] = findIsolatedLabels(Lf,leftLabel);
Lf(vertcat(ccIsolatedLeft{:})) = rightLabel;
% find isolated bodies and replace for right lung
[ccIsolatedRight,~] = findIsolatedLabels(Lf,rightLabel);
Lf(vertcat(ccIsolatedRight{:})) = leftLabel;
% replace any remaining isolated bodies as background
[ccIsolatedLeftRemainder,~] = findIsolatedLabels(Lf,leftLabel);
[ccIsolatedRightRemainder,~] = findIsolatedLabels(Lf,rightLabel);
Lf(vertcat(ccIsolatedLeftRemainder{:})) = backLabel;
Lf(vertcat(ccIsolatedRightRemainder{:})) = backLabel;
end
```

B2.1.7.5.1 findIsolatedLabels

```
function [ccSmall,ccLarge] = findIsolatedLabels(Lf,targetLabel)
%% find isolated connected components in binary map and return indices
% Jason Der
% March 10, 2022
arguments
  Lf
  targetLabel
end
% find connected components for target labels
```

```
ccStruct = bwconncomp(Lf == targetLabel,8);
nCC = ccStruct.NumObjects;
% find the frame (index) of each body using implicit expansion
[nRow,nCol,nFrames] = size(Lf);
ccFirstIdx = cellfun(@(x) x(1),ccStruct.PixelIdxList);
ccFrame = ceil(ccFirstIdx/(nRow*nCol));
% alternative method to find frame index
  % frameLimits = nRow*nCol:nRow*nCol:nRow*nCol*nFrames';
  % [~,ccFrame] = max(ccFirstIdx <= frameLimits(:),[],1);
% find frames with mulitple bodies
[frameCount,frameIdxList] = groupcounts(ccFrame');
frameMultipleCC = frameCount > 1;
idxFrames = frameIdxList(frameMultipleCC); % equivalent to
find(frameMultipleCC)
ccMultiple = ismember(ccFrame,idxFrames);
multipleFrame = ccFrame(ccMultiple);
% find number of pixels per body in frames with multiple connected components
multipleSize = cellfun('length',ccStruct.PixelIdxList(ccMultiple));
% cell for pixel indices of large and small bodies
ccLarge = cell(nFrames,1);
ccLarge(~frameMultipleCC) = ccStruct.PixelIdxList(~ccMultiple);
ccSmall = cell(nCC-nFrames,1);
ccSmallIdx = 0; % start counter
% find pixel indices of large and small bodies
% search frames with multiple connected components
for iFrame = 1:length(idxFrames)
  % connected components to check
  iFrameIdx = idxFrames(iFrame); % frame to check
  multipleIdx = find(multipleFrame==iFrameIdx); % indices for list of
multipleCC
  iFrameSizes = multipleSize(multipleIdx); % number of pixels for each cc to
check
  idx = find(ismember(ccFrame,iFrameIdx)); % cc indices
  % large body indices
  [~,multipleIdxMax] = max(iFrameSizes);
  % large body pixel list
  ccLarge(idxFrames(iFrame)) = ccStruct.PixelIdxList(idx(multipleIdxMax));
  % small body indices
  smallIdx = find(multipleIdx);
  smallIdx(multipleIdxMax) = [];
  % cc small body indices, based on relationship between counter and group
length
  idxChange = length(idx)-1; % number of bodies per frame - 1
  ccSmallIdx = ccSmallIdx + idxChange; % last index in ccSmall to assign too,
update each iteration
  ccSmallArray = ccSmallIdx - idxChange + 1:ccSmallIdx; % indices in ccSmall
  % small body pixel list
  ccSmall(ccSmallArray) = ccStruct.PixelIdxList(idx(smallIdx));
end
end
```

B2.2 Point Cloud Processing

The segmented depth maps were deprojected into point clouds using the follow functions. The deprojectLungs function performs deprojection and some data management. The vertices2PointCloud function performs the post-processing steps such as box average filtering and more data management to convert the deprojected points into point cloud objects. The depth, color, and label data from the images were transferred to the point clouds.

B2.2.1 deprojectLungs

```
function [verticesMatFilename] =
deprojectLungs(streamsMatFilename,segmentationMatFilename,D)
%% deproject lung XYZ and RGB
% Jason Der
% October 30, 2021 updated March 4, 2022 to mirror XYZ about YZ
arguments
  streamsMatFilename
  segmentationMatFilename
  D = 0.38 % depth threshold / distance of EVLP floor to depth origin
end
% load variables
load(streamsMatFilename,'colorImg','depthMap','intrinsics','depthIntrinsics')
;
load(segmentationMatFilename,'tform','Lf','depthMask');
if ~exist('tform','var')
  trans = [0 0 D];
  rot = [-1 0 0;0 1 0;0 0 -1];
  tform = rigid3d(rot,trans);
end
% deproject lung XYZ and RGB data
if ~exist('intrinsics','var')
  intrinsics = depthIntrinsics;
end
[XYZ] = deprojectDepthMap(depthMask.*depthMap,intrinsics);
XYZ = transformPointsForward(tform,XYZ);
XYZ = [-XYZ(:,1),XYZ(:,2:3)];
spatialLimits = [min(XYZ(:,1)),max(XYZ(:,1));
                 min(XYZ(:,2)),max(XYZ(:,2));
                 min(XYZ(:,3)),max(XYZ(:,3))];
XYZ = formatVertices(XYZ,depthMask); % default is cell per frame
% if color image have been segmented
if exist('colorImg','var')
  RGB = formatColorImg(colorImg,depthMask); % default is cell per frame
end
% if regions have been segmented
if exist('Lf','var')
  R = formatLabels(Lf,depthMask);
end
% save results to mat file
[file,name,ext] = fileparts(streamsMatFilename);
verticesMatFilename = fullfile(file,[name 'Vertices' ext]);
```

```
save(verticesMatFilename,'XYZ','spatialLimits','tform','-v7.3');
% if regions have been segmented
if exist('R','var')
  save(verticesMatFilename,'R','-append');
end
if exist('RGB','var')
  save(verticesMatFilename,'RGB','-append');
end
```

B2.2.1.1 deprojectDepthMap

```
function [vertices] = deprojectDepthMap(depthMap,intrinsics)
%% convert depth map[s] into 3D points using pinhole model and inverse Brown-
Conrady distortion model
% Inputs
  % depthMap - depth map[s] with [R,C,N] dimensions
  % intrinsics - ASV intrinsics
    % ppx - physical to pixel scaling in y direction
    % ppy - physical to pixel scaling in x direction
    % fx - focal length in x direction
    % fy - focal length in y direction
% Intermediates
  % ind - linear indices of nonzero depth values
  % sz - size of depthMap = [R,C,N]
  % X - x coordinates in pixels of nonzero depth values
  % Y - y coordinates in pixels of nonzero depth values
  % Z - nonzero depth values
  % x - x coordinates in physical dimensions on image plane (not depth
scaled)
  % y - y coordinates in physical dimensions on image plane (not depth
scaled)
  % ux - x coordinates undistorted (not depth scaled)
  % uy - y coordinates undistorted (not depth scaled)
  % f - a brown conrady distortion coefficient
  % r2 - a brown conrady distortion coefficient
% Outputs
  % vertices - XYZ 3D points in physical space in [N,3] list
% Notes
  % size() provides XY dimensions in opposite order of intrinsics
  % Jason Der
  % October 30, 2021
arguments
  depthMap
  intrinsics
end
% preprocessing
ind = find(depthMap);
sz = size(depthMap);
[Y,X,~] = ind2sub(sz,ind); % FLIP b/c row = y and column = x
Z = depthMap(ind);
clear depthMap
% calculate x and y coorinates
x = (X - intrinsics.ppx) / intrinsics.fx;
y = (Y - intrinsics.ppy) / intrinsics.fy;
% inverse brown conrady model
r2 = x.*x + y.*y;
```

```
f = 1 + intrinsics.coeffs(1) .* r2 + intrinsics.coeffs(2) .* r2.*r2 +
intrinsics.coeffs(5) .* r2.*r2.*r2;
ux = x .* f + 2 * intrinsics.coeffs(3) .* x .* y + intrinsics.coeffs(4) .*
(r2 + 2 .* x .* x);
uy = y .* f + 2 * intrinsics.coeffs(4) .* x .* y + intrinsics.coeffs(3) .*
(r2 + 2 .* y .* y);
% scale x and y coordinates
clear x y f r Y X ind
% vertices = single([Z .* uy,Z .* ux,Z]); <---------- BIG MISTAKE ---------
vertices = single([Z .* ux,Z .* uy,Z]);
end
```

B2.2.1.2 formatVertices

```
function [formatted] = formatVertices(vertices,bw,format)
%% convert [N,3] list of vertices (XYZ) into desired format while separating
list into frames
% Steps
  % determine which frames each vertices point belongs to
  % format into desired format: {cell, pointCloud stack, or ndSparse}
% Inputs
  % vertices - XYZ points in [N,3] list
  % bw - black_white (binary images) image format (including zeros) of
vertices
  % format - string that defines desired format using switch case
% Intermediates
  % ind - linear indices of nonzero pixels in bw
  % bwSize - dimension size of bw [~,~,nFrames]
  % nFrames - number of image frames
  % frame - frame coordinates for nonzero pixels in bw
  % 'x'Frame - index of frame 'x' for looping through frames
  % 'x'FrameChg - indices of when vertices change frames
  % 'x'FramePoints - indices of vertices that are part of frame 'x'Frame
% Outputs
  % formatted - vertices formatted into desired format
% Notes
  % requires ndSparse on FileExchange for ndSparse format option
  % https://www.mathworks.com/matlabcentral/answers/36563-reshaping-2d-
matrix-into-3d-specific-ordering
  % Jason Der
  % November 3, 2021
arguments
  vertices
  bw
  format = 'cell' % 'cell','ndSparse',or 'pc'
end
% determine vertices' dimensions as an image
ind = find(bw);
bwSize = size(bw);
try nFrames = bwSize(3); catch nFrames = 1; end
[~,~,frame] = ind2sub(bwSize,ind);
% format vertices
switch format
  case 'cell'
    % initialize cell array
    formatted = cell(nFrames,1);
    % find when points change frames
```

```
    iFrameChg = [0;find(diff(frame));length(frame)];
    for iFrame = 1:length(iFrameChg)-1
      iFramePoints = iFrameChg(iFrame)+1:iFrameChg(iFrame+1);
      formatted{iFrame} = vertices(iFramePoints,:);
    end
  case 'pc'
    % initialize pointCloud stack
    formatted(1:nFrames,1) = pointCloud([0,0,0]);
    % find when points change frames
    jFrameChg = [0;find(diff(frame));length(frame)];
    for jFrame = 1:nFrames
      jFramePoints = jFrameChg(jFrame)+1:jFrameChg(jFrame+1);
      formatted(jFrame) = pointCloud(vertices(jFramePoints,:));
    end
  case 'ndSparse' % reshape list into sparse frames
    formatted =
ndSparse(permute(reshape(vertices',[3,numel(bw)/nFrames,nFrames]),[2,1,3]));
  otherwise % error message
    disp('invalid format\n');
end
end
```

B2.2.1.3 formatColorImg

```
function [RGB] = formatColorImg(colorImg,depthMask,format)
%% format RGB data into list, matching vertices
% Jason Der
% November 3, 2021
% https://www.mathworks.com/matlabcentral/answers/36563-reshaping-2d-matrix-
into-3d-specific-ordering
% ndSparse option needs work
arguments
  colorImg
  depthMask
  format = 'cell'
end

%% colorImg dimensions and segmentation
  sz = size(colorImg);
  n = sz(1)*sz(2);

%% format segmented colorImg
  switch format
    case 'cell'
      RGB = cell(sz(4),1);
      for i = 1:sz(4)
        colorFrame =
colorImg(:,:,:,i).*repmat(uint8(depthMask(:,:,i)),[1,1,3]);
        ind = find(depthMask(:,:,i));
        RGB{i} = reshape(colorFrame([ind;ind+n;ind+2*n]),[],3);
      end
%     case 'ndSparse'
%       RGB =
ndSparse(permute(reshape(colorSeg',[3,n/sz(4),sz(4)]),[2,1,3]));
    otherwise
      disp('invalid format\n');
  end
```

```
end
```

B2.2.1.4 formatLabels

```
function [R] = formatLabels(L,depthMask)
%% format labelling images into vertices list for point cloud
% replace formatRegions and formatPointCloudIntensityRegion
% just use integer labels in L, use depthMask to identify valid pixels
% Jason Der
% March 4, 2022
arguments
  L
  depthMask
end
% initializations
[~,~,numFrames] = size(depthMask);
R = cell(numFrames,1);
for iFrame = 1:numFrames
% assign region labels to cell as list
labels = L(:,:,iFrame);
R{iFrame} = labels(depthMask(:,:,iFrame));
end
end
```

B2.2.2 vertices2PointCloud

```
function [pointCloudMatFilename] =
vertices2PointCloud(streamsMatFilename,verticesMatFilename,gridStep)
%% prepare XYZ, and associated data (RGB and Regions) for surface
interpolation and integration
% removes invalid points, noise, and ROI outliers
% ROI removal is optional
% Jason Der
% November 7, 2021
% March 4, 2022 added pcdownsample and removed ROI
arguments
  streamsMatFilename
  verticesMatFilename
  gridStep = 0.005
end
% load variables
load(verticesMatFilename,'XYZ','RGB','R');
if ~exist('R','var')
  R = [];
end
% create pointCloud array
pointCloudRGBDI = createPointCloudRGBDI(XYZ,RGB,R);
clear XYZ RGB R
% filter pointCloud array
nFrames = length(pointCloudRGBDI);
pc(1:nFrames,1) = pointCloud([0 0 0]);
for ii = 1:nFrames
  pc(ii) = pcdownsample(pointCloudRGBDI(ii),'gridAverage',gridStep);
  pc(ii) = pcdenoise(pc(ii),'NumNeighbors',10,'Threshold',3);
end
% spatial limits of all valid points
pcSpatialLimits = spatialLimitsPointCloudArray(pc);
```

```
% save data
[file,name,ext] = fileparts(streamsMatFilename);
pointCloudMatFilename = fullfile(file,[name 'PC' ext]);
save(pointCloudMatFilename,'pc','pointCloudRGBDI','pcSpatialLimits','-v7.3');
end
```

B2.2.2.1 createPointCloudRGBDI

```
function [pointCloudMatFilename] =
vertices2PointCloud(streamsMatFilename,verticesMatFilename,gridStep)
%% prepare XYZ, and associated data (RGB and Regions) for surface
interpolation and integration
% removes invalid points, noise, and ROI outliers
% ROI removal is optional
% Jason Der
% November 7, 2021
% March 4, 2022 added pcdownsample and removed ROI
arguments
  streamsMatFilename
  verticesMatFilename
  gridStep = 0.005
end
% load variables
load(verticesMatFilename,'XYZ','RGB','R');
if ~exist('R','var')
  R = [];
end
% create pointCloud array
pointCloudRGBDI = createPointCloudRGBDI(XYZ,RGB,R);
clear XYZ RGB R
% filter pointCloud array
nFrames = length(pointCloudRGBDI);
pc(1:nFrames,1) = pointCloud([0 0 0]);
for ii = 1:nFrames
  pc(ii) = pcdownsample(pointCloudRGBDI(ii),'gridAverage',gridStep);
  pc(ii) = pcdenoise(pc(ii),'NumNeighbors',10,'Threshold',3);
end
% spatial limits of all valid points
pcSpatialLimits = spatialLimitsPointCloudArray(pc);
% save data
[file,name,ext] = fileparts(streamsMatFilename);
pointCloudMatFilename = fullfile(file,[name 'PC' ext]);
save(pointCloudMatFilename,'pc','pointCloudRGBDI','pcSpatialLimits','-v7.3');
end
```

B2.2.2.2 spatialLimitsPointCloudArray

```
function [spatialLimits] = spatialLimitsPointCloudArray(pointCloud)
%% find spatial limits of entire pointCloud array
% Jason Der
% November 14, 2021
arguments
  pointCloud
end
% limits of array elements
X = cell2mat(arrayfun(@(pc) pc.XLimits,pointCloud,'UniformOutput',false));
Y = cell2mat(arrayfun(@(pc) pc.YLimits,pointCloud,'UniformOutput',false));
Z = cell2mat(arrayfun(@(pc) pc.ZLimits,pointCloud,'UniformOutput',false));
```

```
% limits of array
spatialLimits = [min(X(:,1)),max(X(:,2));
                 min(Y(:,1)),max(Y(:,2));
                 min(Z(:,1)),max(Z(:,2))];
end
```

B3 Surface Reconstruction

The point clouds were used to reconstruct the surface of the experiment lungs using the following functions. The interpolateSurface function reconstructed the surface using the 3D coordinates from the point clouds. The interpolateSurfaceColor interpolated the surface map color from the point clouds. The interpolateSurfaceRegion function interpolated the region labels for the left and right lung from the point clouds to the surface map.

B3.1 interpolateSurface

```
function [surfaceMatFilename] =
interpolateSurface(streamsMatFilename,pointCloudMatFilename)
%% interpolate vertices data into surface image
% use primarily griddata() with different interpolation methods
% outputs images for each at query points defined by X and Y
% Jason Der
% developed October 30, 2021 to November 8, 2021
arguments
  streamsMatFilename
  pointCloudMatFilename
end
% load variables
load(pointCloudMatFilename,'pc','pcSpatialLimits');
% surface grid
[X,Y] = interpolationGrid(pcSpatialLimits);
X = double(X);
Y = double(Y);
% initialization
n = length(pc);
[r,c] = size(X);
Z = zeros(r,c,n);
BW = false(r,c,n);
% interpolate at grid points
for ii = 1:n
  % frame XYZ coordinates
  x = double(pc(ii).Location(:,1));
  y = double(pc(ii).Location(:,2));
  z = double(pc(ii).Location(:,3));
  % alpha shape
  shp = alphaShape(x,y,'HoleThreshold',20);
  BW(:,:,ii) = inShape(shp,X,Y);
  % interpolant
  F = scatteredInterpolant(x,y,z,'natural','none');
  % interpolate z coordinates
  Z(:,:,ii) = F(X,Y);
end
```

```
% post-process
Zs = filterSurface(Z);
[DT,validElements,~] = meshSurfaceImg(BW);
% save variables
[file,name,ext] = fileparts(streamsMatFilename);
surfaceMatFilename = fullfile(file,[name 'Surface' ext]);
save(surfaceMatFilename,'BW','Zs','Z','X','Y','DT','validElements','-v7.3');
end
```

B3.1.1 filterSurface

```
function [Zs] = filterSurface(Z,nhood)
%% post process surface
% Jason Der
% December 2, 2021
arguments
  Z
  nhood = [5 5]
end
% initialize
[r,c,n] = size(Z);
Zs = zeros(r,c,n);
% smooth
for ii = 1:n
  [Zs(:,:,ii),~] = wiener2(Z(:,:,ii),nhood);
end
% fill missing
Zs(isnan(Zs)) = 0;
Zs = imfill(Zs,conndef(2,'maximal'));
Zs(Zs == 0) = nan;
end
```

B3.1.2 meshSurfaceImg

```
function [DT,validElements,validNodes] = meshSurfaceImg(Z)
%% create mesh for uniform image
% Jason Der
% September 9, 2021
% made to replace mesh_surfaceImg
arguments
  Z
end
% create general DT for each image
[r,c,n] = size(Z);
[x,y] = meshgrid(1:r,1:c); % X and Y mesh assumed same size as Z
DT = delaunay(x,y); % mesh connectivity
validNodes = reshape(and(Z ~= 0,~isnan(Z)),r*c,n,1); % node is neither 0 or
Nan
validElements = false(size(DT,1),n);
% identify valid elements and nodes for each image in DT
for i = 1:n % ideally vectorize loop
  ind = find(validNodes(:,i));
  validElements(:,i) = all(ismember(DT,ind),2);
end
end
```

B3.2 interpolateSurfaceColor

```
function [surfaceMatFilename] =
interpolateSurfaceColor(surfaceMatFilename,pointCloudMatFilename)
%% interpolate vertices data into surface image
% use primarily griddata() with different interpolation methods
% outputs images for each at query points defined by X and Y
% Jason Der
% developed October 30, 2021 to November 8, 2021
% updated March 4, 2022 to use scattered interpolant + replacing samples
arguments
  surfaceMatFilename
  pointCloudMatFilename
end
% load variables
load(pointCloudMatFilename,'pc','pcSpatialLimits');
% surface grid
[X,Y] = interpolationGrid(pcSpatialLimits);
X = double(X);
Y = double(Y);
% initialization
nFrames = length(pc);
[r,c] = size(X);
C = zeros(r,c,3,nFrames);
% interpolate at grid points
for iFrame = 1:nFrames
  % sample data fo
  xii = double(pc(iFrame).Location(:,1));
  yii = double(pc(iFrame).Location(:,2));
  cii = double(pc(iFrame).Color);
  % interpolate nearest neighbour for RGB color channels
  for jChannel = 1:3
    if jChannel == 1
      % generate scattered interpolant object
      F = scatteredInterpolant(xii,yii,cii(:,jChannel),'nearest','none');
    else
      % change sample values
      F.Values = cii(:,jChannel);
    end
    C(:,:,jChannel,iFrame) = reshape(F(X,Y),r,c);
  end
end
C = uint8(C);
% save variables
save(surfaceMatFilename,'C','-append');
end
```

B3.2.1 interpolationGrid

```
function [X,Y] = interpolationGrid(spatialLimits,nBins)
%% create X and Y grid for interpolation using meshgrid
% determine evenly spaced sample locations using spatial limits and grid
dimensions
% Steps
  % determine distance between sample points based on limits and number of
points
  % determine coordinates of sample points
```

```
% Inputs
  % spatialLimits - pointCloud XYZ limits
  % nBins - dimensions of interpolation grid in x and y
% Intermediates
  % xWidth and yWidth - distance between sample points in x and y coordinates
  % xSample and ySample - x and y coordinates of sample points
% Outputs
  % X and Y - x and y coordinates of sample points as matrices
% Notes
  % has known problem with non-square nBins input when interpolating
  % problem might be caused by this function
  % Jason Der
  % November 7, 2021
arguments
  spatialLimits (3,2) {mustBeFinite}
  nBins (1,2) {mustBeInteger} = [200,200]
end
% distance between sample points
xWidth = (spatialLimits(1,2) - spatialLimits(1,1))/(nBins(1));
yWidth = (spatialLimits(2,2) - spatialLimits(2,1))/(nBins(2));
% sample points as array
xSample = spatialLimits(1,1) + xWidth/2 : xWidth : spatialLimits(1,2) -
xWidth/2;
ySample = spatialLimits(2,1) + yWidth/2 : yWidth : spatialLimits(2,2) -
yWidth/2;
[X,Y] = meshgrid(xSample,ySample);
end
```

B3.3 interpolateSurfaceRegion

```
function [surfaceMatFilename] =
interpolateSurfaceRegion(surfaceMatFilename,pointCloudMatFilename)
%% interpolate vertices data into surface image
% use primarily griddata() with different interpolation methods
% outputs images for each at query points defined by X and Y
% Jason Der
% developed October 30, 2021 to November 8, 2021
arguments
  surfaceMatFilename
  pointCloudMatFilename
end
% load variables
load(pointCloudMatFilename,'pc','pcSpatialLimits');
% surface grid
[X,Y] = interpolationGrid(pcSpatialLimits);
X = double(X);
Y = double(Y);
% initialization
nFrames = length(pc);
[r,c] = size(X);
R = zeros(r,c,nFrames);
% interpolate at grid points
for iFrame = 1:nFrames
  xii = double(pc(iFrame).Location(:,1));
  yii = double(pc(iFrame).Location(:,2));
  rii = double(pc(iFrame).Intensity);
  F = scatteredInterpolant(xii,yii,rii,'nearest','none');
```

```
  R(:,:,iFrame) = reshape(F(X,Y),r,c);
end
% convert to uint8
Lz = uint8(R);
% save variables
save(surfaceMatFilename,'Lz','-append');
end
```

B4 Measurement

The surface maps were used to measure plethysmography metrics using the measurePorcineLung function or a similar function for the other experiment lungs. Most measurements such as tidal volume were derived a displacement signal obtained from surface integration using the Divergence Theorem. Also, the estimateSurfaceParams function was used to measure regional measurements of surface tidal displacement. The create_avg_cycle function was used to obtain a plethysmography and regional respiratory cycle average displacement. The compare_to_avg_cycle function measured deviation from the respiratory cycle average.

B4.1 measurePorcineLung

```
function [metricsMatFilename] =
measurePorcineLung(streamsMatFilename,surfaceMatFilename,VSCaptureMatFilename
)
%% estimate tidal volume and flow measurements from uniformly interpolated
surface
% Jason Der
% November 8, 2021, last updated December 8, 2021
arguments
  streamsMatFilename
  surfaceMatFilename
  VSCaptureMatFilename
end
% load variables
load(streamsMatFilename,'t','timestamp');
load(surfaceMatFilename,'X','Y','Zs','DT','BW');
load(VSCaptureMatFilename,'measureTableSync');
Time = t;
Timestamp = timestamp;
% volume
[Distension,~] = integrateSurfaceGrid(X,Y,Zs.*BW,DT);
Distension = 1000000*Distension; % m^3 to mL
DistensionSmooth = preprocessSignal(Distension,0.2,30);
% local extremas and tidal volume
[Local extremas,Local extremaTF] = findLocal extremas(DistensionSmooth);
[IEpts] = findLocal extremaChangePoints(Local extremas,Local extremaTF);
[VTi,VTe] = estimateTidalVolume(DistensionSmooth,IEpts);
% flow rate and flow time
[FlowRate,FlowTime,FlowFilterDelay] = flowRate(DistensionSmooth,Time);
% find dynamic compliance
PIP = measureTableSync.Peak_inspiratory_pressure(IEpts);
```

```
PEEP = measureTableSync.Positive_endexpiratory_pressure(IEpts);
[Cdyni,Cdyne] = estimateDynamicCompliance(VTi,VTe,PIP,PEEP,IEpts);
% volume and flow parameters
[PTF,PTFpts] = findCyclePeaks(FlowRate,IEpts,'cycle');
[PTIF,PTIFpts] = findCyclePeaks(FlowRate,IEpts,'inhale');
[PTEF,PTEFpts] = findCyclePeaks(FlowRate,IEpts,'exhale');
[tPTIF,tPTEF] = findTimePeakTidalFlow(Time,IEpts,PTIFpts,PTEFpts);
[tTIF50,tTEF50] = findTimeTidalFlow50(Time,DistensionSmooth,IEpts);
[TIF50,TEF50,IE50] = findTidalFlow50(FlowRate,Time,tTIF50,tTEF50);
[tI,tE,tTot,tITot,tIE,RR] = tidalBreathingTiming(Time,IEpts);
% convert parameters per cycle to an array the length of the signal
[~,CycleLabels]  = paramCycles2Array(IEpts,IEpts,length(Time));
[~,FlowCycleLabels]  = paramCycles2Array(PTFpts,PTFpts,length(Time));
% distension table
distensionFlowTable = table(...
  Timestamp,Time,Distension,DistensionSmooth,...
  VTi,VTe,Cdyni,Cdyne,CycleLabels,...
  FlowRate,FlowCycleLabels);
% param table
paramTable = table(PTF,PTIF,PTEF,PTFpts,PTIFpts,PTEFpts,...
  tPTIF,tPTEF,tTIF50,tTEF50,TIF50,TEF50,IE50,tI,tE,tTot,tITot,tIE,RR);
% tidal table
VTi = VTi(IEpts); Cdyni = Cdyni(IEpts);
VTe = VTe(IEpts); Cdyne = Cdyne(IEpts);
pts =
struct('IEpts',IEpts,'PTFpts',PTFpts,'PTIFpts',PTIFpts,'PTEFpts',PTEFpts);
tidalTable = table(VTi,VTe,Cdyni,Cdyne);
% vscapture tidal table
VTi = measureTableSync.Tidal_volume(IEpts); VTe = VTi;
Cdyni = measureTableSync.Dynamic_compliance(IEpts); Cdyne = Cdyni;
vsTidalTable = table(VTi,VTe,Cdyni,Cdyne);
% save variables to MAT file
[file,name,ext] = fileparts(streamsMatFilename);
metricsMatFilename = fullfile(file,[name 'Metrics' ext]);
save(metricsMatFilename,'distensionFlowTable','paramTable','vsTidalTable','ti
dalTable','pts','FlowTime','FlowFilterDelay');
end
```

B4.1.1 estimate_volume

```
function [volume,elementVolume] = estimate_volume(points,DT)
%% Jason Der
% January 2, 2021
% estimates the volume under the surface tri-mesh
% inputs: MATLAB delaunay triangulation object
% outputs: total volume, volume, and volume per element, elementVolume
% updated september 15, 2021 arugment names

%% math references
% same method as pneumacare
% similar method here: https://rosenzweig.io/blog/hilariously-fast-volume-
computation-with-the-divergence-theorem.html
% engineering paper, similar algorithm:
http://chenlab.ece.cornell.edu/Publication/Cha/icip01_Cha.pdf
% gauss theorem (divergence = flux) with vector field F(x,y,z) = z, div*F = 1
% only in z unit direction, only surfaces with projection onto xy plane
% simplifies gauss theorem to volume = flux from z vector
```

```
%% function argument validation
  arguments
    points (:,3) double {mustBeReal}
    DT (:,3) double {mustBeNonnegative,mustBeReal} = delaunay(points(1:2,:))
  end

%% initialization
  elementVolume = zeros(length(DT),1);
  volume = 0;

%% calculate volume via gauss theorem
  % per triangluar element in mesh
  for i = 1:length(DT)
    % triangluar element vertices
    p0 = points(DT(i,1),:);
    p1 = points(DT(i,2),:);
    p2 = points(DT(i,3),:);
    % triangluar element z components
    z0 = points(DT(i,1),3);
    z1 = points(DT(i,2),3);
    z2 = points(DT(i,3),3);
    % surface function, parameter partial derivatives
    r1 = p1 - p0; r2 = p2 - p0;
    % triangle integral
    integral = cross(r1,r2)*[0;0;(z0+z1+z2)];
    % volume of triangular element i
    elementVolume(i) = integral/6;
    % summate integral to volume
    volume = volume + integral;
  end
  volume = volume/6;

end
```

B4.1.2 estimateTidalVolume

```
function [vT1,vT2,vT] = estimateTidalVolume(v,IEpts)
%% find tidal volume
% does not know which tidal volume is inspiratory or expiratory
% just provides them in order of occurence, odd first, even second
% Jason Der
% September 27, 2021 updated from estimate_tidal_volume from Feb 18, 2021
% updated November 9, 2021
arguments
  v % volume signal
  IEpts % indices of local extrema points
end
% tidal volume
vT = diff(v(IEpts));
% initialize tidal volume arrays
vT1 = zeros(length(v),1);
vT2 = zeros(length(v),1);
% loop through cycles
for ii = 1:2:length(vT)-1
  % cycle indices
```

```
  cycleInd = IEpts(ii):IEpts(ii+2);
  % assign tidal volumes to matching cycles in array
  vT1(cycleInd) = abs(vT(ii));
  vT2(cycleInd) = abs(vT(ii+1));
end
end
```

B4.1.3 integrateSurfaceGrid

```
function [v,vDT] = integrateSurfaceGrid(X,Y,Z,DT)
%% integrate surface mesh for volume curve
% Jason Der
% October 30, 2021 - November 9, 2021
arguments
  X (:,:,1) % surface meshgrid x coordinates
  Y (:,:,1) % surface meshgrid y coordinates
  Z (:,:,:,1) % surface meshgrid z coordinates
  DT (:,:,1) % surface meshgrid connectivity
end
% initialize variables
x = X(:);
y = Y(:);
[r,c,n] = size(Z);
v = zeros(n,1);
vDT = cell(n,1);
% integrate surface for volume
for ii = 1:n
  z = reshape(Z(:,:,ii),r*c,1);
  zValid = find(~isnan(z));
  dtValid = all(ismember(DT,zValid),2);
  [v(ii),vDT{ii}] = estimate_volume([x,y,z],DT(dtValid,:));
end
end
```

B4.1.4 preprocessSignal

```
function [smooth,outliers,denoise] =
preprocessSignal(signal,Fpass,Fs,windowSize,nsigma)
%% pre-process signal
% Jason Der
% December 3, 2021
arguments
  signal
  Fpass = 0.2 % band pass frequency,
  % respiratory rate 8 breaths/min = 0.13...Hz -> 0.2 Hz with buffer
  Fs = 30 % sampling frequency (30fps)
  windowSize = min(0.01*length(signal),Fs) % default to 1% of length, max at
Fs
  nsigma = 3
end
% denoise
denoise = lowpass(signal,Fpass,Fs);
% replace outliers
windowSize = ceil(windowSize); % must be integer
outliers = hampel(denoise,windowSize,nsigma);
% window size must be odd for sgolayfilt
if ~mod(windowSize,2)
  windowSize = windowSize+1;
```

```
end
% smooth using savitzsky-golay method
smooth = sgolayfilt(outliers,3,windowSize);
end
```

B4.1.5 flowRate

```
function [signalDT,timeDT,filterDelay] =
flowRate(signal,time,Fpass,Fstop,Forder)
%% find first derivative of signal using differential filter
% use to find analogous 'flow rate' as the first derivative of distension
% also find transient filter delay
% Jason Der
% August 26, 2021
% NOTES:
  % based on MATLAB tutorial for differentiator filter
  % https://www.mathworks.com/help/signal/ug/take-derivatives-of-a-
signal.html
arguments
  signal
  time
  Fpass = 1 % band pass frequency
  Fstop = 1.2 % stop frequency
  Forder = 10 % filter order
end
% sampling frequency and time interval
Fs = 1/(time(2)-time(1));
dt = time(2)-time(1);
% design derivative filter
d = designfilt('differentiatorfir',...
  'FilterOrder',Forder, ...
  'PassbandFrequency',Fpass,...
  'StopbandFrequency',Fstop, ...
  'SampleRate',Fs);
% differentiate
signalDT = filter(d,signal)/dt;
filterDelay = mean(grpdelay(d)); % transient delay from filter
timeDT = time(1:end - filterDelay);
% signalDT(1:filterDelay) = []; % remove delay
end
```

B4.1.6 findLocal extremas

```
function [local extremas,pksTrsTF] = findLocal
extremas(signal,minHeight,minDistance,method)
%% find and sort local extremas
% can use findLargestLocal extremas or findChangePointLocal extremas to
filter
% local extrema selection, however, just smoothing signal is a simple
solution
% Jason Der
% August 27, 2021
%% change log
% updated September 16, 2021
  % determine arguments and one local extrema per group
  % intended for noisy ventilation signals: typical signal shape is flat-
sharp 'S'-plateau-immedidate drop-repeat
  % intended to find start and end of inspiratory and expiratory phases
```

```
% formatted November 11, 2021
% updated November 15, 2021
  % returns viepts starting with inspiratory / trough, ignores expiratory
start
% updated November 30, 2021
  % use islocalmin and islocalmax instead of findpeaks
  % findpeaks does not work for data with trend
  % separated into findLocal extremas and findBreathLocal extremas
%% program
arguments
  signal
  minHeight = []
  minDistance = 0
  method = 'findpeaks'
end
% remove outliers, and detrend
% fillSignal = filloutliers(signal,'nearest');
dt = detrend(signal);
% estimate MinHeight
if isempty(minHeight) && all(method=='findpeaks')
  minHeight = mean(dt);
end
% estimate peaks and troughs
switch method
  case 'findpeaks'
    % findpeaks
    [~,pks] =
findpeaks(dt,'MinPeakHeight',minHeight,'MinPeakDistance',minDistance);
    [~,trs] = findpeaks(-dt,'MinPeakHeight',-
minHeight,'MinPeakDistance',minDistance);
  case 'islocal'
    % islocal
    pks = find(islocalmax(dt));
    trs = find(islocalmin(dt));
end
% sorted inspiratory-expiratory local extrema point indices
[local extremas,sortOrder] = sort([pks;trs]); % linear
% logical array identifying local extremas as peaks or troughs
pksTrsTF = [true(length(pks),1);false(length(trs),1)]; % logical
pksTrsTF = pksTrsTF(sortOrder); % logical sorted
end
```

B4.1.7 findLocal extremaChangePoints

```
function [vieptsPaired,pksInd,trsInd,viepts] = findLocal
extremaChangePoints(local extremas,pksTrsTF)
%% find filter local extremas for breath start-end local extremas
% remove repeated peaks and troughs
  % identify which local extrema points neighbor the opposite phase
  % where inspiratory changes to expiratory, visa versa
  % use to ignore insipratory to inspiratory, visa versa
  % ie. local extremas = [i i e e i i] -> [i i e e i i e] -> [0 1 0 1 0 1]
  % end padded to identify inspiratory at end
% Jason Der
% November 30, 2021
arguments
  local extremas % linear indices for local extremas
```

```
  pksTrsTF logical % identifies local extrema [i e] -> [1 0]
end
% find change point between peaks and troughs
ieInd = [diff(pksTrsTF)~=0;true]; % logical, diff is end padded
% final local extrema indices
viepts = local extremas(ieInd); % linear
% truncate first and/ or last indice[s] if expiratory to create complete IE
pairs
vieptsPaired = viepts;
ieTF = pksTrsTF(ieInd); % logical, T -> expiratory, F -> inspiratory
% truncate first indice if expiratory
if ieTF(1)
  vieptsPaired = vieptsPaired(2:end);
end
% truncate last indice if expiratory
if ieTF(end)
  vieptsPaired = vieptsPaired(1:end-1); % linear
end
% signal value at inhale-exhale points
pksInd = local extremas(and(ieInd,pksTrsTF)); % linear
trsInd = local extremas(and(ieInd,~pksTrsTF)); % linear
end
```

B4.1.8 findChangePeaks

```
function [Peaks,PeaksInd] = findCyclePeaks(signal,IEpts,type)
%% find signal max values per cycle and indices
%
% STEPS
%
% initialize outputs
% find cycle range indices
% loop through inhales and find max pressure
%
% INPUTS
%
% pressure = pressure signal
% IEpts = linear indices of both inhale-exhale points in paired order
% type =
%
% INTERMEDIATES
%
% cycleStartInd =
% cycleEndInd =
% numCycles =
%
% OUTPUTS
%
% Peaks =
% PeaksInd =
%
% EXAMPLE
%
% NOTES
%
% Jason Der
% February 21, 2022
```

```
arguments
  signal
  IEpts
  type = 'cycle'
end
% initialize outputs based on desired usage
switch type
  case 'inhale' % use to find PIP from pressure
    cycleStartInd = IEpts(1:2:end);
    cycleEndInd = IEpts(2:2:end);
  case 'exhale' % use to find PEEP from pressure
    cycleStartInd = IEpts(2:2:end);
    cycleEndInd = IEpts(3:2:end);
    signal = -signal;
  case 'cycle' % use to find peak flow rate
    cycleStartInd = IEpts(1:2:end);
    cycleEndInd = IEpts(3:2:end);
  otherwise
    disp('invalid type variable\n');
    return
end
% initialize outputs
numCycles = floor((length(IEpts)-1)/2);
Peaks = zeros(numCycles,1);
PeaksInd = zeros(numCycles,1);
% loop through cycles
for xCycle = 1:numCycles
  % find inhale range
  cycleIndices = cycleStartInd(xCycle):cycleEndInd(xCycle);
  % find max signal and indices within cycle
  cycleSignal = (signal(cycleIndices));
  [~,ind] = max(cycleSignal,[],'all','linear','omitnan');
  PeaksInd(xCycle) = ind + cycleStartInd(xCycle);
  Peaks(xCycle) = signal(PeaksInd(xCycle));
end
end
```

B4.1.9 findTimePeakTidalFlow

```
function [tPTIF,tPTEF] = findTimePeakTidalFlow(Time,IEpts,PTIFpts,PTEFpts)
%% find tidal breathing parameter 'time to peak tidal flow'
% find for both inspiratory and expiratory phases
% parameter is time between start of phase and peak flow
% the code was modified from tidalBreathingFlow()
% Jason Der
% November 15, 2021
% updated February 22, 2022
arguments
  Time
  IEpts
  PTIFpts
  PTEFpts
end
% time at start of cycle
cycleStartTime = Time(IEpts);
% time at peak flow
peakTidalInspiratoryFlowTime = Time(PTIFpts);
```

```
peakTidalExpiratoryFlowTime = Time(PTEFpts);
% time to peak tidal flow
tPTIF = peakTidalInspiratoryFlowTime - cycleStartTime(1:2:end-1);
tPTEF = peakTidalExpiratoryFlowTime - cycleStartTime(2:2:end);
end
```

B4.1.10 findTimeTidalFlow50

```
function [tTIF50,tTEF50] = findTimeTidalFlow50(time,volume,IEpts)
%% find the tidal breathing parameters TIF50,TEF50,IE50,tTIF50,tTEF50
% TIF50,TEF50 is the flow when volume is at 50% of the tidal volume
% IE50 is a ratio between TIF50 and TEF50
% tTIF50, tTEF50 is the time between the start of a breath to TIF50,TEF50
% the code was modified from tidalBreathingFlow()
% Jason Der
% November 15, 2021
arguments
  time
  volume
  IEpts
end
% tidal volume
% 50% tidal volume per cycle
% linear indices of 50% tidal volume per cycle <----- interpolation
% flow rate at 50% tidal volume per cycle <---- interpolation
tidalVolumeIE = diff(volume(IEpts));
volumeIE50 = tidalVolumeIE/2 + volume(IEpts(1:end-1)); % volume at 50% tidal
volume
timeTidalVolumeIE50 = zeros(length(volumeIE50),1); % pre-allocate
% interpolate time at volume when 50% tidal volume
for i = 1:length(volumeIE50)
  indicesRange = IEpts(i) : IEpts(i+1);
  timeRange = time(indicesRange);
  volumeRange = volume(indicesRange);
  volumeSearch = volumeIE50(i);
  [~,search] = min(abs(volumeRange-volumeSearch));
  timeTidalVolumeIE50(i) = timeRange(search);
end
% time of TEF50, TIF50, IE50 for plotting
tTIF50 = timeTidalVolumeIE50(1:2:end);
tTEF50 = timeTidalVolumeIE50(2:2:end);
end
```

B4.1.11 findTidalFlow50

```
function [TIF50,TEF50,IE50] = findTidalFlow50(flow,time,tTIF50,tTEF50)
%% find tidal breathing parameters TIF50,TEF50,IE50
% parameters are the flow when volume is at 50% of tidal volume
% code is modified from estimateTidalBreathing
% Jason Der
% November 15, 2021
arguments
  flow
  time
  tTIF50
  tTEF50
end
try
```

```
% could upsample flow and time then use method 'nearest'
% interpolate flow at IE50 times
TEF50 = interp1(time,flow,tTIF50,'spline');
TIF50 = interp1(time,flow,tTEF50,'spline');
catch
% alternative method
[~,idx] = min(abs(time-tTIF50'));
TIF50 = flow(idx);
[~,idx] = min(abs(time-tTEF50'));
TEF50 = flow(idx);
end
% find IE50 as ratio between TIF50 and TEF50
IE50 = TIF50 ./ TEF50;
end
```

B4.1.12 tidalBreathingTiming

```
function [tI,tE,tTot,tITot,tIE,RR] = tidalBreathingTiming(time,IEPts)
%% calculate tidal breathing parameters derived from timing indices
% STEPS
% perform basic equations
% INPUTS
% time = time signal
% IEPts = inhale-exhale points (local extrema points in distension signal)
% INTERMEDIATES
% numBreath = number of breaths
% OUTPUTS
% tI = inspiratory time
% tE = expiratory time
% tTot = total time per breath
% tITot = inspiratory / total time ratio
% tIE = inspiratory / expiratory time ratio
% RR = respiratory rate
% EXAMPLE
% time = repelem([1 0],10,1);
% IEPts = [1;find(diff(time));length(time)];
% [tI,tE,tTot,tITot,tIE,RR] = tidalBreathingTiming(time,IEPts);
% NOTES
% read : Reference equations for tidal breathing parameters using structured
light plethysmography
% read : Tidal breathing patterns derived from structured light
plethysmography in COPD patients compared with healthy subjects
% read : Tidal breathing parameters measured using structured light
plethysmography in healthy children and those with asthma before and after
bronchodilator
% Jason Der
% July 1, 2021 from tidal_breathing_parameters
% Rewritten on August 25, 2021 and February 22, 2021
arguments
  time (:,1)
  IEPts (:,1)
end
% find timing characteristics
timeIE = diff(time(IEPts));
tI = timeIE(1:2:end);
tE = timeIE(2:2:end);
tTot = diff(time(IEPts(1:2:end)));
```

```
numBreath = length(tTot);
tIE = tI(1:numBreath)./tE(1:end);
tITot = tI(1:numBreath)./tTot;
RR = 60./tTot;
end
```

B4.2 estimateSurfaceParams

```
function [metricsMatFilename] =
estimateSurfaceParams(surfaceMatFilename,metricsMatFilename,VSCaptureMatFilen
ame)
%% estimate surface measurements
% Jason Der
% November 8, 2021
arguments
  surfaceMatFilename
  metricsMatFilename
  VSCaptureMatFilename = []
end
% load variables
load(surfaceMatFilename,'X','Y','Zs');
load(metricsMatFilename,'distensionTable','pts');
t = distensionTable.Time;
IEpts = pts.IEpts;
% surface metrics
[SamplePoints,nSamplePoints] = findBaselineSamplePoints(t,IEpts);
[BaselineSurface] = calculateSurfaceBaseline(Zs,SamplePoints,nSamplePoints);
[DeviationSurface] =
calculateSurfaceDeviation(Zs,BaselineSurface,SamplePoints,nSamplePoints);
[VTiSurface,VTeSurface] = calculateSurfaceTidalVolume(Zs,IEpts);
if ~isempty(VSCaptureMatFilename)
  load(VSCaptureMatFilename,'measureTableSync');
  PIP = measureTableSync.('Peak_inspiratory_pressure')(IEpts(2:2:end));
  PEEP =
measureTableSync.('Positive_endexpiratory_pressure')(IEpts(2:2:end));
  [CdyniSurface,CdyneSurface] =
calculateSurfaceCompliance(VTiSurface,VTeSurface,PIP,PEEP);
  save(metricsMatFilename,'CdyniSurface','CdyneSurface','-append');
end
% save variables
save(metricsMatFilename,'SamplePoints','nSamplePoints','X','Y',...
  'BaselineSurface','DeviationSurface','VTiSurface','VTeSurface','-append');
end
```

B4.2.1 findBaselineSamplePoints

```
function [samplePoints,nSamplePoints] =
findBaselineSamplePoints(t,viepts,nSamplePoints)
%% find surface grid baseline sample points
% find time interval between sample points
% equal number of sample points between inspiratory and expiratory phases
% sample points should be always taken at local extrema points
% search for the nearest time element to ideal sample point times
% take indices of these nearest time elements
% Jason Der
% November 11, 2021
arguments
  t (:,1) % sample data matching time
  viepts (:,1) % linear indices identifying local extrema points
  nSamplePoints (1,1) {mustBeNonnegative,mustBeInteger} = 19 % must be odd
end
% ensure nSamplePoints is odd
if ~rem(nSamplePoints,2) % if even
```

```
  nSamplePoints = nSamplePoints + 1; % add one, making it odd
end
% number of intervals between sample points per (inspiratory or expiratory)
phase
nPhaseIntervals = (nSamplePoints - 1)/2;
% time length of (inspiratory or expiratory) phases
phaseTimeIntervals = diff(t(viepts));
% time interval between sample points per (inspiratory or expiratory) phase
samplePointTimeIntervals = phaseTimeIntervals / nPhaseIntervals;
% repeat nrptDelta m times
samplePointTime = [samplePointTimeIntervals(1);
cumsum(repelem(samplePointTimeIntervals,nPhaseIntervals)) +
samplePointTimeIntervals(1)];
% indices of closet points to ideal sample points
samplePoints = dsearchn(t,samplePointTime);
% replicate every nSamplePoints so each cycle is complete
samplePoints = sort([samplePoints;samplePoints(nSamplePoints:nSamplePoints-
1:end-1)]);
end
```

B4.2.2 calculateSurfaceBaseline

```
function [zBaseline] = calculateSurfaceBaseline(Z,samplePoints,nSamplePoints)
%% create average cycle, averaged reference point data set from samples
% Jason Der
% November 11, 2021
arguments
  Z % surface grid image stack
  samplePoints % linear indices
  nSamplePoints = 19 % number of sample points
end
zSample = Z(:,:,samplePoints);
[r,c] = size(Z(:,:,1));
missingSamplePoints = nSamplePoints -
rem(length(samplePoints),nSamplePoints); % ... from last breath
if missingSamplePoints > 0
  % pad reference image data with NaN images
  zSample = cat(3,zSample,nan(r,c,missingSamplePoints));
end
% reshape 4D array into 3D array and average over reference images
zBaseline = squeeze(mean(reshape(zSample,r,c,nSamplePoints,[]),4,'omitnan'));
end
```

B4.2.3 calculateSurfaceDeviation

```
function [zDeviation] =
calculateSurfaceDeviation(Z,zBaseline,samplePoints,nSamplePoints)
%% find deviation from surface baseline
% Jason Der
% November 11, 2021
arguments
  Z double {mustBeNumeric(Z)} % data to compare against average cycle
  zBaseline double {mustBeNumeric(zBaseline)} % average cycle dataset of
length nrptsN, either vector or 3D array
  samplePoints (:,1) double {mustBeNumeric(samplePoints)} % linear indices of
reference points in reference cycle
  nSamplePoints (1,1) {mustBeNonnegative,mustBeInteger} = 19
end
```

```
nSampleBreaths = length(samplePoints)/nSamplePoints;
zSample = Z(:,:,samplePoints);
zBaselineReplicate = repmat(zBaseline,1,1,nSampleBreaths);
BW = (zSample~=0) & (zBaselineReplicate~=0); % where both are true
zDeviation = BW .* (zSample - zBaselineReplicate);
end
```

B4.2.4 calculateSurfaceTidalVolume

```
function [VTiSurface,VTeSurface,VTSurface] =
calculateSurfaceTidalVolume(Z,IEpts)
%% find tidal surface distension
% Jason Der
% July 22, 2022
arguments
  Z % surface
  IEpts % distension local extrema indices
end
% define pairs
if rem(length(IEpts),2) == 0 % if even
  endInd = length(IEpts);
else % if odd
  endInd = length(IEpts)-1;
end
% surface tidal volume where surfaces are common
VTSurface = diff(Z(:,:,IEpts),1,3);
% surface tidal volume inspiratory-expiratory
VTiSurface = VTSurface(:,:,1:2:endInd);
VTeSurface = VTSurface(:,:,2:2:endInd);
end
```

B4.2.5 calculateSurfaceCompliance

```
function [Cdyni Surface,Cdyne Surface] =
calculateSurfaceCompliance(VTiSurface,VTeSurface,PIP,PEEP)
%% scale surface tidal volume by PIP and PEEP for surface dynamic compliance
% Jason Der
% July 22, 2022
arguments
  VTiSurface
  VTeSurface
  PIP
  PEEP
end
% length of arrays
nVTi = size(VTiSurface,3);
nVTe = size(VTeSurface,3);
nPIP = length(PIP);
nPEEP = length(PEEP);
if (nVTi == nVTe) && (nPIP == nPEEP)
  % initialize
  Cdyni Surface = zeros(size(VTiSurface));
  Cdyne Surface = zeros(size(VTeSurface));
  % pressure differential
  deltaP = PIP - PEEP;
  % loop through frames
  for iFrame = 1:nVTi
    Cdyni Surface(:,:,iFrame) = VTiSurface(:,:,iFrame) ./ deltaP(iFrame);
```

```
    C_dyneSurface(:,:,iFrame) = VTeSurface(:,:,iFrame) ./ deltaP(iFrame);
  end
else
  disp('error');
end
```

B4.3 create_avg_cycle

```
function [avg_cycle,ind] =
create_avg_cycle(sampleData,sampleTime,iepts,nrpts)
%% create average cycle, averaged reference point data set from samples
% Jason Der
% June 4, 2021
% does not have validation protection against m x n where m && n ~= 1
% updated September 27, 2021 to separate oultiers replacement and correct 'm'

%% function argument validation
  arguments
    sampleData % sample data, either vector or 3D array for images
    sampleTime (:,1) % sample data matching time
    iepts (:,1) % linear indices identifying local extrema points
    nrpts (1,1) {mustBeNonnegative(nrpts)} = 19
      % number of reference points per cycle (including iepts)
      % must be odd, if even then round up
  end

%% round up even nrpts
  % if nrpts even
  if rem(nrpts,2) == 0
    % increase nrpts by one to be odd
    nrpts = nrpts + 1;
  end

%% find reference points time vector
  % number of spaces between reference points, between local extrema points
  n = (nrpts - 1)/2;
  % time at local extrema points
  ieTime = sampleTime(iepts);
  % time difference between neighbouring local extrema points
  ieTimeDiff = diff(ieTime);
  % time separation for reference points between local extrema points
  nrptDelta = ieTimeDiff / n;
  % repeat nrptDelta m times
  nrptTime = [ ieTime(1); cumsum(repelem(nrptDelta,n)) + ieTime(1)];

%% find points nearest to reference points in sample time
  % search for nearest sample data for each reference point
  ind = dsearchn(sampleTime,nrptTime);
  % replicate every nrpt indices in ind so each cycle is complete
  ind = sort([ind;ind(nrpts:nrpts-1:end-1)]);

%% find average for each reference point in sample data
  % last cycle incomplete, need m points to be complete
%   m = rem(length(ind),nrpts);
  m = ceil(length(ind)/nrpts)*nrpts - length(ind);
  % if sample data is a matrix (vector)
```

```
  if ismatrix(sampleData)
    % take reference point data at ind
    refData = sampleData(ind);
    % if m is greater than zero
    if m > 0
      % pad reference point data with NaN
      refData = [refData;repelem(nan,m,1)];
    end
    % reshape into matrix to average over reference points
    avg_cycle = mean(reshape(refData,nrpts,[]),2,'omitnan');

  % if sample data 3D array
  elseif ~ismatrix(sampleData)
    % take reference images at ind
    refData = sampleData(:,:,ind);
    % size of images
    [r,c] = size(sampleData(:,:,1));
    % if m is greater than zero
    if m > 0
      % pad reference image data with NaN images
      % refData = cat(3,refData,repmat(nan(r,c),m,1));
      refData = cat(3,refData,nan(r,c,m));
    end
    % reshape 4D array into 3D array and average over reference images
    avg_cycle = squeeze(mean(reshape(refData,r,c,nrpts,[]),4,'omitnan'));
  end

end
```

B4.4 compare_to_avg_cycle

```
function [diff_cycle] =
compare_to_avg_cycle(sampleData,avg_cycle,ind,sampleTime,iepts)
%% compare data to average cycle set
% Jason Der
% June 4, 2021
% updated September 27, 2021 to correct 'm'

%% function argument validation
  arguments
    sampleData double {mustBeNumeric(sampleData)} % data to compare against
average cycle
    avg_cycle double {mustBeNumeric(avg_cycle)} % average cycle dataset of
length nrptsN, either vector or 3D array
    ind (:,1) double {mustBeNumeric(ind)} = [] % linear indices of reference
points in reference cycle
    sampleTime (:,1) double {mustBeNonnegative(sampleTime)}  = [] % matching
time vector to data
    iepts (:,1) double {mustBeNonnegative(iepts)}  = [] % linear indices for
local extrema points in data and time
  end

%% prepare data
  % replace Nan with zero
  sampleData(isnan(sampleData)) = 0;
  avg_cycle(isnan(avg_cycle)) = 0;
```

```
%% initializations
  % if vector
  if ismatrix(sampleData)
    % number of normalized reference points
    nrpts = length(avg_cycle);
  % else is 3D array (images)
  else
    % number of normalized reference points
    [~,~,nrpts] = size(avg_cycle);
  end

%% shortcut if ind given
  % if ind is empty
  if isempty(ind)

  %% find reference points time vector
    % number of spaces between reference points, between local extrema points
    n = (nrpts - 1)/2;
    % time at local extrema points
    ieTime = sampleTime(iepts);
    % time difference between neighbouring local extrema points
    ieTimeDiff = diff(ieTime);
    % time separation for reference points between local extrema points
    nrptDelta = ieTimeDiff / n;
    % repeat nrptDelta m times
    nrptTime = [ ieTime(1); cumsum(repelem(nrptDelta,n)) + ieTime(1)];

  %% find points nearest to reference points in sample time
    % search for nearest sample data for each reference point
    ind = dsearchn(sampleTime,nrptTime);
    % replicate every nrpt indices in ind so each cycle is complete
    ind = sort([ind;ind(nrpts:nrpts-1:end-1)]);

  end

%% compare sample data to reference cycle
  % number of avg_cycle replications to get same length as refData
%   k = (length(ind)-m)/nrpts;
  k = ceil(length(ind)/nrpts);
  % last cycle incomplete, need m points to be complete
%   m = rem(length(ind),nrpts);
  m = abs(length(ind) - nrpts*k)-1;
  % if sample data is a matrix (vector)
  if ismatrix(sampleData)
    % take reference point data at ind
    refData = sampleData(ind);
    % replicate avg_cycle to same length as refData
    avgData = repmat(avg_cycle,k,1);
    % if m is greater than zero
    if m > 0
      % remove last m entries
      avgData(end-m:end) = [];
    end
    % AND operation between avgData and refData (where both nonzero)
```

```
    nonzero = refData & avgData;
    % subtract data from reference cycle, and use nonzero mask to filter
    diff_cycle = nonzero .* (refData - avgData);

  % if sample data 3D array
  elseif ~ismatrix(sampleData)
    % take reference images at ind
    refData = sampleData(:,:,ind);
    % replicate avg_cycle to same length as refData, k times
    avgData = repmat(avg_cycle,1,1,k);
    % if m is greater than zero
    if m > 0
      % remove last m entries
      avgData(:,:,end-m:end) = [];
    end
    % AND operation between avg_cycle and refData (where both nonzero)
    nonzero = refData & avgData;
    % subtract avgData from refData and use nonzero mask to filter
    diff_cycle = nonzero .* (refData - avgData);
  end

end
```

B4.5 estimateRegionAsychrony

```
function [tidalMetricsMatFilename] =
estimateRegionAsynchrony(tidalMetricsMatFilename)
%% estimate region tidal breathing asynchrony
% Jason Der
% November 11, 2021
arguments
  tidalMetricsMatFilename
end
% load variables
load(tidalMetricsMatFilename,'lungParams','regionParams');
% find combintations for asyncrhony
nRegions = length(regionParams);
if nRegions >= 2
  regionComs = nchoosek(1:nRegions,2);
else
  fprintf('zero or one regions detected \n');
  return;
end
% loop through combinations
for ii = 1:size(regionComs,2)
  % select combination of regions
  v1 = regionParams(regionComs(ii,1)).Volume;
  v2 = regionParams(regionComs(ii,2)).Volume;
  local extremas = regionParams(regionComs(ii,1)).VolumeIEIndicesPaired;
  % estimate asynchrony between regions
  [phaseAngleDegrees] = paradoxicalBreathingAsynchrony(v1,v2,local extremas);
  [IP] = paradoxicalBreathingTiming(lungParams,regionParams(ii));
  % save variables to struct variable
  asynchronyParams(ii) =
struct('SelectedRegions',regionComs(ii,:),'PhaseAngle',phaseAngleDegrees,'Ins
piratoryParadoxTime',IP);
```

```
end
% save to MAT file
save(tidalMetricsMatFilename,'asynchronyParams');
end
```

B4.5.1 paradoxicalBreathingAsynchrony

```
function [phaseAngleDegrees] = paradoxicalBreathingAsynchrony(v1,v2,local
extremas)
%% calculate signal asynchrony
% Jason Der
% August 25, 2021, debugged Sept 2, 2021
% NOTES:
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0216641
% November 16, 2021
  % outputs complex numeb
  arguments
    v1 % volume signal of region 1
    v2 % volume signal of region 2
    local extremas % local extrema indices of volume signal of region 1
  end

%% determine inspiratory-expiratory order
  check = v1(local extremas(2)) > v1(local extremas(1));
  if check % inhale start
    in = 1; ex = 2; % indices of first inhale and exhale start local extrema
  elseif ~check % exhale start
    in = 2; ex = 3; % indices of first inhale and exhale start local extrema
  end
  nCycles = length(local extremas(in:2:end))-1;

%% lung wall tidal volumes
  % find tidal volume of region 1 per breath
  tidalVolumeIE1 = diff(v1(local extremas(in:in+2*nCycles)));
  s = max(abs(reshape(tidalVolumeIE1,2,[])))';
  % find 50% tidal volume of region 2 for region 1 breath
  volume50 = zeros(nCycles,1);
  for i = 1:nCycles
    indSearchRange = local extremas(in+((i-1)*2)):local extremas(in+(2*i));
    maxRange = max(v2(indSearchRange));
    deltaV2 = maxRange - min(v2(indSearchRange));
    volume50(i) = maxRange - deltaV2/2;
  end

%% degree of asynchrony
  m = zeros(nCycles,1);
  inspiratoryIntercept = zeros(nCycles,1);
  expiratoryIntercept = zeros(nCycles,1);
  for j = 1:nCycles
    % start and end indices of breath k
    inspiratoryIndRange = local extremas(in+((j-1)*2)):local
extremas(in+1+((j-1)*2));
    expiratoryIndRange = local extremas(ex+((j-1)*2)):local
extremas(ex+1+((j-1)*2));
    % inspiratory and expiratory volume data to interpolate
    [~,in1] = unique(v1(inspiratoryIndRange),'stable');
```

```
    [~,in2] = unique(v2(inspiratoryIndRange),'stable');
    [~,ex1] = unique(v1(expiratoryIndRange),'stable');
    [~,ex2] = unique(v2(expiratoryIndRange),'stable');
    % unique pairs
    inspiratoryVolumeRange1 = v1(inspiratoryIndRange(intersect(in1,in2)));
    inspiratoryVolumeRange2 = v2(inspiratoryIndRange(intersect(in1,in2)));
    expiratoryVolumeRange1 = v1(expiratoryIndRange(intersect(ex1,ex2)));
    expiratoryVolumeRange2 = v2(expiratoryIndRange(intersect(ex1,ex2)));
    % interpolate for volume in region 1 when volume in region 2 = 50% of
tidal volume
    inspiratoryIntercept(j) =
interp1(inspiratoryVolumeRange2,inspiratoryVolumeRange1,volume50(j),'spline')
;
    expiratoryIntercept(j) =
interp1(expiratoryVolumeRange2,expiratoryVolumeRange1,volume50(j),'spline');
    m(j) = inspiratoryIntercept(j) - expiratoryIntercept(j);
  end
  phaseAngleDegrees = asind(abs(m)./s);

end
```

B4.5.2 paradoxicalBreathingTiming

```
function [IP] = paradoxicalBreathingTiming(lungParams,regionParams)
%% find inspiratory paradox time for each region
% Jason Der
% created Sept 2, 2021, found tidal breathing parameters, lung and regions,
within script
% updated November 11, 2021 to recieve structs for lung and region tidal
breathing parameters
arguments
  lungParams struct % entire lung tidal breathing parameters
  regionParams struct % struct region tidal breathing parameters
end
% inspiratory times
regionInTime = regionParams.InspiratoryTime;
lungInTime = lungParams.InspiratoryTime;
% length of inspiratory times
nRegions = length(regionInTime);
nLung = length(lungInTime);
% correct if number of inspiratory times match
if nRegions ~= nLung
  % find start time of each breath
  regionIETime = regionParams.Time(regionParams.VolumeIEIndicesPaired);
  lungIETime = lungParams.Time(lungParams.VolumeIEIndicesPaired);
  % select matching breath inspiratory times
  regionInTime = [];
  lungInTime = [];
end
% inspiratory paradox time
IP = 100 * (lungInTime - regionInTime)./lungInTime;
end
```

Appendix C Ventilator Test Lung Data Acquisition Settings

Table C.1 Ventilation test lung experiment ASV calibration intrinsic parameters

Parameters	Left Infrared	Right Infrared	RGB
Resolution	1280 × 800	1280 × 800	1920 × 1080
Focal Length	$\begin{bmatrix}638.065002\\637.815002\end{bmatrix}$	$\begin{bmatrix}641.018982\\640.336975\end{bmatrix}$	$\begin{bmatrix}1379.920044\\1380.84997\end{bmatrix}$
Principal Point	$\begin{bmatrix}639.348999\\400.669006\end{bmatrix}$	$\begin{bmatrix}634.036011\\404.847992\end{bmatrix}$	$\begin{bmatrix}953.692017\\545.323975\end{bmatrix}$
Distortion	$\begin{bmatrix}-0.056679\\0.063950\\0.000533\\-0.000361\\-0.020585\end{bmatrix}$	$\begin{bmatrix}-0.057750\\0.065349\\-0.000502\\-0.001087\\-0.020886\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}$

Table C.2 Ventilation test lung experiment ASV calibration extrinsic parameters

Sensor	Rotation to Left Infrared	Translation to Left Infrared
Left Infrared	n/a	n/a
Right Infrared	$\begin{bmatrix}0.999896 & -0.001195 & -0.014369\\0.001178 & 0.999999 & -0.001225\\0.014371 & 0.001208 & 0.999896\end{bmatrix}$	$\begin{bmatrix}-50.158493\\-0.049040\\-0.190886\end{bmatrix}$
RGB	$\begin{bmatrix}0.999781 & 0.018581 & -0.009628\\-0.018548 & 0.999822 & 0.003496\\0.009691 & -0.003316 & 0.999948\end{bmatrix}$	$\begin{bmatrix}14.404202\\-0.117552\\0.546147\end{bmatrix}$

Table C.3 Ventilation test lung experiment depth quality metrics

Metrics	Depth Quality
Resolution	848 × 480
Frame Rate	30
Region of Interest	40 %
Fill Rate	100%
Z-Accuracy	-0.03 %
Plane Fit RMS Error	0.33 %
Subpixel RMS Error	0.00 pixels

Appendix D Porcine Lung Data Acquisition Settings

Table D.1 Porcine test lung experiment ASV calibration intrinsic parameters

Parameters	Left Infrared	Right Infrared	RGB
Resolution	1280 × 800	1280 × 800	1920 × 1080
Focal Length	$\begin{bmatrix}638.065002\\637.815002\end{bmatrix}$	$\begin{bmatrix}641.018982\\640.336975\end{bmatrix}$	$\begin{bmatrix}1379.920044\\1380.84997\end{bmatrix}$
Principal Point	$\begin{bmatrix}639.348999\\400.669006\end{bmatrix}$	$\begin{bmatrix}634.036011\\404.847992\end{bmatrix}$	$\begin{bmatrix}953.692017\\545.323975\end{bmatrix}$
Distortion	$\begin{bmatrix}-0.056679\\0.063950\\0.000533\\-0.000361\\-0.020585\end{bmatrix}$	$\begin{bmatrix}-0.057750\\0.065349\\-0.000502\\-0.001087\\-0.020886\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}$

Table D.2 Porcine test lung experiment ASV calibration extrinsic parameters

Sensor	Rotation to Left Infrared	Translation to Left Infrared
Left Infrared	n/a	n/a
Right Infrared	$\begin{bmatrix}0.999896 & -0.001195 & -0.014369\\0.001178 & 0.999999 & -0.001225\\0.014371 & 0.001208 & 0.999896\end{bmatrix}$	$\begin{bmatrix}-50.158493\\-0.049040\\-0.190886\end{bmatrix}$
RGB	$\begin{bmatrix}0.999781 & 0.018581 & -0.009628\\-0.018548 & 0.999822 & 0.003496\\0.009691 & -0.003316 & 0.999948\end{bmatrix}$	$\begin{bmatrix}14.404202\\-0.117552\\0.546147\end{bmatrix}$

Table D.3 Porcine test lung experiment depth quality metrics

Metrics	Depth Quality
Resolution	848 × 480
Frame Rate	30
Region of Interest	40 %
Fill Rate	100%
Z-Accuracy	-0.03 %
Plane Fit RMS Error	0.33 %
Subpixel RMS Error	0.00 pixels