Computing Velocity of Multiple Objects in Sequences of Images With an Application In Water-Based Bitumen Extraction Process

by

Mahdi Shooshtari

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Image-based analysis of bitumen extraction process can provide the oil companies with useful information that can be used to assess their performance in retrieving bitumen from the oil sands. In this analysis, several slurry images are taken during the extraction process, and then image processing techniques are used to extract the information and operating metrics from the images. Computing the velocity of floating objects, including bitumen droplets and sand particles, is one of the most important operating metrics. This operating metric gives us information about bitumen retrieving performance in addition to more understanding about the floating objects and their movement; yet up to our knowledge, there is not an evaluated automated method to detect and track the floating objects in slurry images successfully and measure their speed in the real-time. To address this problem, we have developed algorithms to (1) detect bitumen droplets and sand particles in each image; and (2) track the detected objects in the sequence of images.

For the first step of this project, we evaluated several well-known global and local thresholding and segmentation algorithms and found the method with the best outcome for our images. We also applied tiling and downsampling methods to the images in order to improve the performance of evaluated thresholding and segmentation algorithms in the object detection and/or decreasing running time of the segmentation algorithms. The results indicated that tiling and downsampling decrease the performance in most of the cases. Tiling and downsampling of the images decrease the running time of the segmentation algorithms, but they still are not as fast as thresholding methods. Moreover, we developed a Recursive Iterative Thresholding framework that can be combined with any local or global thresholding algorithm to improve the performance of the original algorithm through detecting a larger number of small and bright objects.

For the second step of the project (i.e. tracking the detected objects in a sequence of images), we implemented and tested two tracking methods: (1) a frame-by-frame tracking algorithm; and (2) a multi-frame tracking method. For the frame-by-frame tracking algorithm, objects of consecutive images were matched and connected to each other by using an assigning algorithm that determines the optimum assignment. In contrast, for the multi-frame tracking method, multiple frames were considered together, and a Flow Network was built by connecting the objects using the weighted edges. Then a greedy approach was used to detect the K-shortest path (KSP) in the Flow Network. Our experiments indicated that finding the KSP in a Flow Network has better performance compared to the frame-by-frame tracking, and it produces the results in a noticeably less running time. We used two State-of-the-Art cutting algorithms and developed two additional novel methods to cut the tracks into smaller tracklets for using in the Flow Network. Our experiments showed that using each tracklet as a node of the Flow Network instead of using each detected object as a node improved the performance of the algorithm.

Acknowledgements

I would first like to thank my thesis adviser Professor Hong Zhang of the Department of Computing Science at the University of Alberta. The door to Prof. Zhang office was always open whenever I ran into a troubled spot or had a question about my research or writing. He consistently allowed this project to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank an expert, and a great friend, who was involved in the validation of this research project: Postdoctoral Fellow of the Department of Computing Science, University of Alberta, Dr. Li He. Without his passionate participation and input, this thesis could not have been successfully conducted.

I would also like to acknowledge Dr. Nilanjan Ray of the Department of Computing Science at the University of Alberta as the second reader of this thesis, and I am gratefully indebted to him for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my parents and to my sister for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author
Mahdi Shooshtari

Table of Contents

List of Tables

List of Figures

List of Abbreviations

List of commonly used abbreviations

2D-ENT	Two Dimensional Entropy
BFB	Between Frames Based
DAG	Directed Acyclic Graph
GB	Graph-based
GUI	Graphical User Interface
KSP	K Shortest Path
MAT	Multistage Adaptive Thresholding
MAX-ENT	Maximum Entropy
MHT	Multiple Hypothesis Tracking
MIN-ERR	Minimum Error
M-SHIFT	Mean Shift
MTT	Multiple Target Tracking
SLIC	Simple Linear Iterative clustering
SRG	Seeded Region Growing
SRM	Statistical Region Merging
TB	Track Based

Chapter 1

Introduction

1.1 Computing Multiple Objects Velocities: Technology and Application

Oil sands or technically called bitumen sands are mixtures of bitumen, sand, clay and water. In order to separate bitumen from mined oil sands ore, there is a water-based procedure that includes crushing mined oil sands ore and mixing them with hot water, steam, and some chemicals to produce oil sands slurry [10]. The slurry is then given to a hydro-transported pipeline in which the majority of bitumen separates from sand particles and attaches to existed air bubbles [10, 23]. Bitumen droplets (either attached to air bubbles or remained attached to sand particles) in addition to pure sand particles form the main objects in the slurry. To collect bitumen droplets and dispose of sand particles, the slurry is then introduced into a Primary Separation Vessel (PSV). In PSV, bitumen droplets that are attached to air bubbles float up and make the *primary bitumen froth*, while the sand particles and the bitumen droplets that are attached to sand particles either sink as underflow and settle at the bottom of PSV, or remain suspended in the middle of the vessel as middlings. These objects are processed separately to recover more bitumen froth (called *secondary bitumen froth*) from them [10, 23]. Any process applied to the oil sands slurry is called the slurry process [10].

Recently an image-based slurry process analyzing system has been invented [10] that is capable of retrieving useful information from bitumen extraction process. It generates system performance metrics for the slurry processing and adjusts different operating parameters in order to produce the optimal performance in retrieving bitumen from the slurry. This system contains a flow tube, one or two cameras, and a computer. The flow tube, which is a colorless cylindrical glass has a slurry inlet, a slurry outlet, and one or two transparent viewing portions. The input of slurry inlet is diverted from the slurry process and it includes either slurry before *primary bitumen froth* separation or the slurry that only contains middlings and settled objects. The slurry is then floated in the flow tube, and cameras that are installed in the viewing portions capture its images. These images are supplied to a computer for further processings. In the computer, different analysis can be applied to the slurry images and different information can be retrieved. This includes bitumen droplet size and shape, bitumen droplet velocity, predicted bitumen recovery rate, sand particle size distribution, sand particle shape, and water turbidity [10]. There are two main processing steps, at least one of which is required in order to extract each of the aforementioned information: **(A)** Detecting objects in each image; and **(B)** Tracking the detected objects in a sequence of slurry images. Computing velocity of the bitumen droplets which is the focus of our study requires both of the above processing steps.

In this project, we processed the images that contain multiple objects most of which are sand particles, and the rest are sand-bitumen combinations. One camera takes several good quality images of the objects as they sink until the operation is over. Some examples of different moments of the process after floating slurry in the flow tube are shown in Figure 1.1. The two main steps to automatically compute the velocity of floating objects include **(A)** detection of as many objects (i.e. bitumen droplets and sand particles) as possible in each image and **(B)** tracking the detected objects (mainly bitumen droplets) in the sequences of images. There are some computational challenges in detection and tracking of objects in slurry images: (1) Most of the objects in the slurry images are very small or bright and they are overlapping. Based on our initial literature review, we found that most of the published segmentation methods are not capable of addressing this challenge properly. (2) The objects rotate during the sinking process and their shape or size and color change as they move. On the other hand, the velocity of different objects is not the same. These conditions make automated tracking of the bitumen droplets in the series of images very challenging.

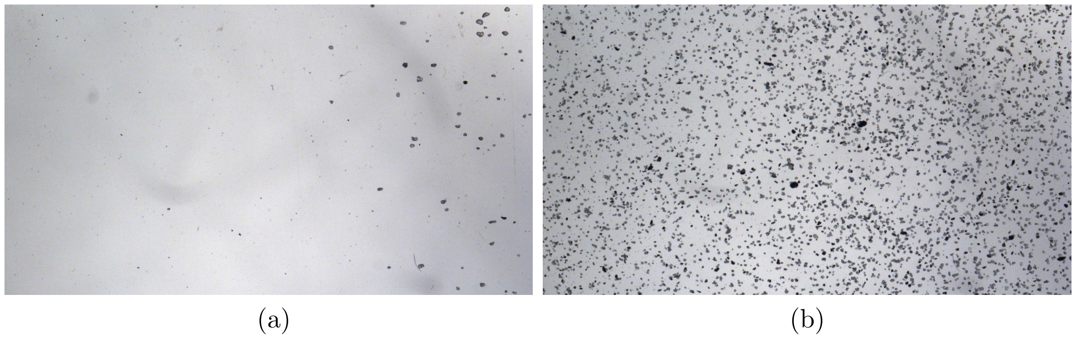


Figure 1.1: Examples of slurry images in the beginning (a) and middle (b) moments of floating slurry in the flow tube. At the beginning moments, there are fewer objects to detect and the images are sparse. After a while, there are hundreds of objects and the images are dense.

1.2 Detecting Objects in Each Image

As shown in Figure 1.1, the images normally contain a large number of small objects with different intensities and shapes. Although some of the sand particles and bitumen droplets are large and dark enough to be detected easily by the majority of the thresholding algorithms, most of the objects do not have these characteristics. Additionally, the images suffer from different illuminations at different parts. This makes the object detection problem even more challenging. On the other hand, it is required to compute the velocity of objects as fast as possible, preferably in the real-time. Considering all of these together, the desired method needs to detect small and bright objects as well as the large or dark objects and needs to be fast enough to run in the real-time.

1.2.1 Image Thresholding

One of the main applications of image segmentation is to detect objects in images. In segmentation methods, the main idea is to divide an image into smaller similar parts in a way that it makes the required object or the Region of Interest (RoI) easily distinguishable from other parts of the image. Image thresholding is a specific category of segmentation algorithms. In thresholding algorithms, the input is normally a gray image in which each pixel has a value in the range of [0, 255]. The output is a binary image. In the output, the pixels that contain Objects or RoI get 1 (which is the white color), while the background pixels get 0 (which is the black color). In order to be consistent with the input images, one can invert the value and color of each pixel of the output image. Deciding on whether a pixel should get 0 or 1 in the output image is made based on a threshold value (T):

$$J(x,y) = \begin{cases} 0, & \text{if } \ I(x,y) < T(x,y) \\ 1, & \text{otherwise} \end{cases} \tag{1.1}$$

where J is the output image, I is the input image, x and y are the current pixel's coordinates and T(x,y) is the threshold value for the pixel in x and y coordinates. In the thresholding algorithms, the main objective is to select the threshold value T(x,y). There have been many efforts to develop appropriate methods to identify the value of the threshold [37, 45].

Thresholding algorithms can be used for different applications such as blood cell or other medical particle tracking methods [7, 36, 33], recognizing text in low quality or ancient document images [31, 25, 50, 12, 34]. In the text recognition applications, thresholding algorithms can benefit from other information such as background estimation, features of the text and/or learning techniques [25, 50, 12].

1.2.2 Global Thresholding

Thresholding algorithms can be divided into two main groups; **(A)** global thresholding; and **(B)** local thresholding. Global thresholding methods find the optimum threshold value for the entire image. In other words, for any pixel, the threshold value T(x,y) is the same (called T). Global thresholding methods decide on the threshold value based on the image features such as intensity histogram, image entropy or clustering the intensity values [47]. Normally cut-off threshold value T divides the bimodal histogram into two separate clusters with unimodal histograms. Under an ideal situation, the histogram of the image intensity has a bimodal distribution. However, this condition does not hold most of the time, meaning that histograms are not bimodal, and therefore, the separated histograms cannot perfectly fit into two unimodal diagrams.

Maximum entropy (MAX-ENT) [15] is a well-known global thresholding method that is based on entropy, and finds a threshold that maximizes the sum of entropy in each of the separated clusters. Minimum error (MIN_ERR), Otsu and Iterative are examples of clustering-based methods [19, 32, 29]. Minimum error (MIN-ERR) [19] assigns the best Gaussian model to each of the unimodal histograms and then computes the sum of the misclassification error. This method finds a threshold that minimizes misclassification error. The Iterative method [32] computes the mean intensity of each class separately and updates the next threshold value iteratively as the mean sum of classes divided by two. This method repeats this procedure until the threshold value does not change substantially. Otsu method [29] identifies a threshold value such that it provides the highest similarity between members of each class through minimizing within class variance, and the highest difference between members of different classes by maximizing between class variance.

1.2.3 Local Thresholding

Global thresholding algorithms do not have good performance when there are shadows, poor illumination or a lot of noise in the images. To address this problem, local thresholding algorithms select a specific threshold value for each pixel according to its spatial properties such as relation with neighbor pixels. Niblack [27] computes the threshold value in each pixel as a function of intensity average and diversity deviation in a window neighborhood. The threshold value for each pixel is computed by the following equation:

$$T(x, y) = \mu(x, y) + K * \sigma(x, y) + C \tag{1.2}$$

where $\mu(x, y)$ is the average intensity of a window with size *b*b* while the center is in coordinates x and y. *T(x,y)* is the computed threshold value for the pixel in x and y, $\sigma(x, y)$ is standard deviation and C is the offset. Width of the window b can be constant as it is in Niblack method or adaptive with regard to more spatial properties. Here K is negative.

Sauvola [34] is an improved version of Niblack that relaxes the effect of standard deviation $\sigma(x, y)$ by the proposed formula:

$$T(x, y) = \mu(x, y) * \left[1 + K(\frac{\sigma(x, y)}{R} - 1)\right] \tag{1.3}$$

where *R=128* and K is a positive number. This approach is suitable for image areas with low contrast. In those cases, $\sigma(x, y)$ is low and $T(x, y)$ is less than $\mu(x, y)$. As a result, the dark regions which are part of the background are detected and removed more easily.

NICK [18] smooths the noise in image neighborhood while applying a local thresholding method similar to Niblack.

$$T(x, y) = \mu(x, y) + K * \sqrt{\frac{\sum_{i=1}^{n} I(x_i, y_i)^2 - \mu(x, y)^2}{N}} \tag{1.4}$$

where $n = b*b$, N is the total number of pixels in the image and $I(x_i, y_i)$ is the intensity of i^{th} pixel in the neighborhood in coordinates x_i and y_i. Recently, Khoshki [17] made NICK algorithm more accurate by iteratively repeating the procedure with different descending window sizes and selecting the objects that were detected as objects in the previous step.

Bernsen [4] is a simpler method in which the threshold is determined as follows:

$$T(x, y) = \begin{cases} 128 & \text{if } \ I_{max}(x, y) - I_{min}(x, y) < \epsilon \\ \frac{I_{max}(x,y) + I_{min}(x,y)}{2} & \text{Otherwise} \end{cases} \tag{1.5}$$

the threshold value for a pixel is only computed when the difference between maximum and minimum intensity $I_{max}(x, y) - I_{min}(x, y)$ in a neighborhood is bigger than a constant number ϵ, otherwise it is in the same class of neighbor pixels. That means if the pixels of the neighborhood have similar intensities and current pixel has an intensity less than 128 (half range in gray scale), it is assigned to the background, otherwise to the foreground.

Two-Dimensional Entropy (2D-ENT) [1], as an entropy based method, maps every pixel in the image into a two-dimensional environment where one axis is the intensity value and the other axis is the intensity average in the neighborhood. This algorithm finds two threshold values (one for each axis) that split the 2D environment into two groups, and it provides the maximum sum of 2D-ENT in the classes.

Multistage Adaptive Thresholding (MAT) algorithm [47] assumes that selecting a proper threshold value is difficult when the intensity of pixels are in a certain range:

$$T(x,y) = \begin{cases} T_{min} & \textit{if } I(x,y) < T_{min} \\ T_{max} & \textit{if } I(x,y) > T_{max} \\ \mu(x,y) + K * \sigma(x,y) & \textit{Otherwise} \end{cases} \tag{1.6}$$

When the intensity of a pixel is less than a predefined small value T_{min} it is assigned to the background and when it is more than predefined big value T_{max} it is considered as an object with no doubt. Selection of T_{min} and T_{max} is one of the main contributions of this paper.

Gatos [12] aims for detecting a text in degraded document images. It first computes a rough estimation of the text by Sauvola Algorithm. Then it computes the background estimation with a function affected by computed foregrounds and finally it detects the desired objects based on the difference between original image and the estimated background image.

Local thresholding algorithms, in general, are time-consuming because they need to compute a threshold value for each of the pixels one by one. To address this problem, some algorithms use integral images [40, 38, 17]. Here the main idea is that for each pixel in an integral image, its value is equal to the sum of all previous pixels that are in left and top position of this pixel including the current one in the original image. By having this integral image, computing average or variance of the intensity for each pixel in a neighborhood with a size of w*w can be done easily. As a result, the processing time is changed from $O(n^2w^2)$ in an n*n image into $O(n^2)$, and it is not dependent on the size of window (w) anymore [38]. We used some of the methods (Niblack, Sauvola and NICK) along with the same idea in Section 2.3 and Section 2.4 for evaluation and comparison.

1.2.4 Image Segmentation

Image segmentation algorithms divide an image into several areas with the most similar pixels. Unlike thresholding methods, output images of image segmentation can have multiple gray scale intensities and each area will have a different label. By segmenting the slurry images in a proper way, sand particles, bitumen droplets, and water can be distinguished from each other.

Seeded region growing (SRG) [3] is a segmentation method that starts with some pixels that are known to be part of the foreground or ROI (as seeds). This method then expands the area around these seeds to find the whole foreground region. The space around each seed will be enlarged until the difference between the intensity of neighbor pixels and the mean of regional intensities is greater than a threshold.

Mean shift (M-SHIFT) [8] maps each pixel in an RGB image into a five-dimensional environment (Red, Green, Blue, X_coordinate, and Y_coordinate) and then tries to shift each point to the most crowded closest position around. Clusters made by this approach create the segments of the image. The same process can be done for the case of gray scale images by shifting each pixel to the nearest most crowded place in a three-dimensional environment (Intensity, X_coordinate, and Y_coordinate).

In graph-based (GB) method [11], pixels of the image are nodes in the graph and edge weights of edges are computed according to a function of dissimilarity between corresponded nodes. This method aims to divide the graph into smaller subgraphs so that each sub-graph (cluster) should contain similar members. As a result, edges between two vertices in the same cluster should have relatively low weights. Members of two different clusters should be different enough from each other. As a result, edges between two vertices in the different cluster should have relatively higher weights. Edges will then be sorted according to their weights incrementally. For each edge in the sorted list, if nodes of the edge are in different classes but they are similar enough (according to measurements), related classes would be merged. There are two ways for making the graph; **(A)** considering edges between each pixel and its eight neighbors which is called Grid Graph or **(B)** map each pixel of the image into a five-dimensional space. For **(B)**, Red, Green, Blue, X_coordinate, and Y_coordinate are the axis. Each pixel has edges with its N nearest neighbors in a specific radius R. This method is called nearest neighbor (NN) graph. In the Grid Graph, the weight of each edge is equal to the intensity difference of related pixels. While in the NN Graph weights are equal to the Euclidean distance between points. in the Grid Graph, S is the minimum size of a segmentation cluster.

In statistical region merging (SRM) [28] each pixel is compared with its four neighbors (with regard to a function of dissimilarity) and four connected pairs are formed. These pairs are sorted incrementally based on their dissimilarity value. From top to bottom of this sorted list, if two nodes of a connected pair are not in the same segment, but they are similar enough (according to a similarity function), two related segments will be merged together. This method needs one variable to be set which is related to the number (and size) of the detected segments.

Superpixel algorithms group adjacent similar pixels into small partitions called superpixels. In generating them, two main purposes are considered; **(A)** adhering as well as possible to the boundaries of objects in the image and **(B)** running fast. Simple linear iterative clustering (SLIC) method for grouping similar pixels is comparable to K-means [2]. Assume having K superpixels is desired. As a result, according to the number of pixels in the image which is N, the image initially is divided into squares with an approximate length of $S = \sqrt{\frac{N}{K}}$ and S cluster centers are found. For each pixel in the image, it is then assigned to the nearest cluster center. Edges of each square should adhere to edges of objects. By increasing K there would be more superpixels but with smaller areas. It will also increase the chance of having accurately boundary fitted superpixels.

1.3 Tracking Objects in a Sequence of Images

In order to compute the speed of objects, we need to track the detected objects in a sequence of images. The computational challenges for addressing this problem are as follows:

- The main features that can be extracted from detected objects are the shape (size), color (brightness) and their location. There are hundreds of detected objects in each image and most of the particles in our images have similar features. This makes it challenging to match the same particles in sequences of images.
- In general, objects appear at the top of the images, sink vertically and after a while disappear at the bottom of the images. This process is very fast and in most of the cases, the lifetime of an object in our images is less than ten frames. A lifetime of super fast objects can be two or three frames.
- Because of having a moderate water flow in the tube, objects do not necessarily move vertically and they also have movements in other directions. Moreover, objects collide with other objects and there is a possibility that they merge together. During the sinking process, merged objects might split from each other later. As a result, the number of objects in consecutive images is not necessarily the same.
- Images have different lighting in different that affect the brightness of objects during their sinking.
- Objects rotate in the water and their observed shapes are not constant.
- Objects do not move at the same speed. Moreover, the speed of an object varies as it moves.

1.3.1 Multiple Object Tracking

Tracking several objects simultaneously, called as Multiple Target Tracking (MTT) is a famous problem that is needed to be solved in different applications. Tracking blood cells for medical usages [20, 22, 24, 13, 14, 35, 9], tracking pedestrians in the images of surveillance cameras [48, 30, 41] or tracking players in team sports to have team analysis [39] are examples of MTT. We are interested mainly in mentioned applications because of objects of those images, blood cells, and people, have some similarities to our bitumen droplets and sand particles. For example, objects can be merged or stayed together for several frames, they have different speeds and also their shape and color are not constant during the movement. Similarly, targets to track in slurry images are bitumen droplets and sand particles (in general we call them as objects) that show similar behaviors.

Many different algorithms are proposed to address MTT problem for different situations. In general, there is not a single best method that performs excellent for all different types of images and objects, and depending on images and object features different algorithms have different performances [7, 24]. Both supervised and unsupervised algorithm for MTT are developed and their performance among different experiments indicates that none of them are absolutely better than others [7]. In our specific application, we are not able to use supervised and machine learning based algorithms because we do not have enough training data for our slurry images and there is not any data set with similar features available. Merging of moving objects together and/or splitting them from each other are other common situations that in most of the MTT cases happen [13]. We mentioned some algorithms that handled these two cases in Section 1.3.2 and Section 1.3.3.

Most of the MTT algorithms can be categorized into one of the two main groups: **(A)** algorithms which assign newly detected objects in each frame to the existed tracks especially by comparing the new objects with the objects of the previous frame; and **(B)** algorithms that find the best possible tracks by taking all objects of all frames into consideration. First group is called frame-by-frame tracking algorithms (more details in Section 1.3.2) and second group is called multi-frame tracking algorithms (more details in Section 1.3.3) [22, 7, 24, 13]. A vast study on different MTT algorithms shows that in general, multi-frame tracking algorithms have better performances than frame-by-frame tracking. However, most of the multi-frame tracking algorithms need more time to run and might not be suitable for real-time tracking processes [7].

1.3.2 Frame-by-frame Tracking Algorithms

Frame-by-frame tracking algorithms are MTT methods that aim to make the tracks by iteratively assigning (i.e. matching) the objects of two consecutive images starting from the first two frames and ending by two last frames of a video. These methods connect the newly detected objects to the existing tracks. The assignment is based on similarity of the objects in successive frames. In our images, similarity can be computed according to features such as shape, color, and positions. General framework of frame-by-frame tracking algorithms mentioned in [5] is shown as a flowchart in Figure 1.2.

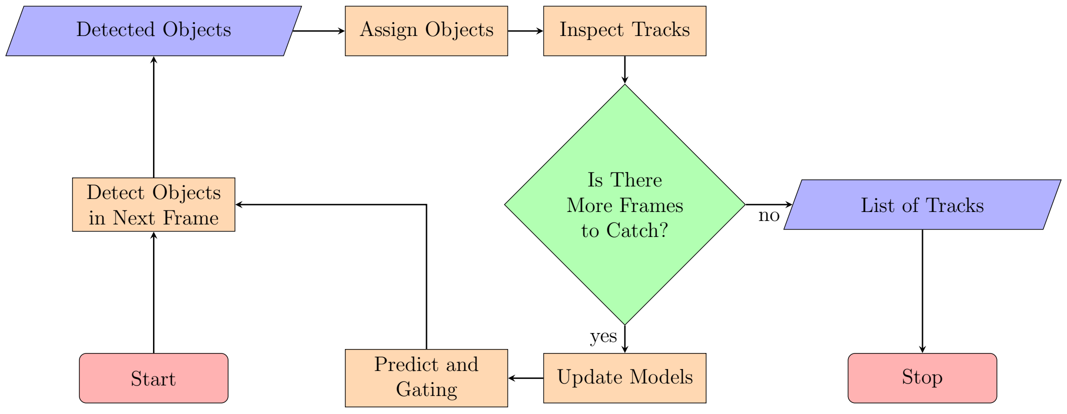


Figure 1.2: General framework of frame-by-frame tracking algorithms.

Although different algorithms use different ways to track the objects, all of them have at least the assigning step of the framework. For example; [9] assigns each object in the current frame to the Local Nearest Neighbor (LNN) object in the next image [7, 21, 33, 35]. Improved methods aim to find a global optimal assigning between objects. Global Nearest Neighbor (GNN) [35] algorithms allocate the objects in a way to get the highest similarity score between connected pairs. Hungarian assigning is also a similar method which finds the least total cost between connected objects [7, 26]. More complicated algorithms are also proposed; using each object's shape as a kernel to find its best match in the next image [21], or in [33] case, fitting the best possible curve around each object by using a fourth order polynomial Gaussian function in order to gain more possible features and information from objects.

Track inspection step is to make sure the connections between objects are correct. This step is also responsible for handling the cases such as disappearing an object for several frames (blinking), and merging or splitting of objects. Moreover, ending a track or starting a new one are two other tasks related to this step. It is not necessarily possible to assign all objects in a frame to all objects in the next frame. Some algorithms ignore the blinking, merging and splitting problems and their focus is on tracking non-conflicting objects [21, 35]. On the other hand, many algorithms are proposed to handle these cases, for example, by using dummy objects (non-existed objects) to make the number of objects in successive frames equal [36, 9]. Linking the short tracks to each other to make complete tracks is also used in [14]. In addition, requesting user input in conflicting situations is another way to handle merging and splitting occurrences [42].

Comparing objects in one frame to all objects in the next frame to find the best possible matches is very time intensive. Only a few features can be extracted per object and since there are a significant number of objects in each image, many objects would be similar to each other. Therefore the chance of false assigning would be higher compared to cases that there are fewer objects to compare with. Shrinking the search window is a good solution for eliminating many of the false candidates [7, 5]. This method called gating and it increases the speed of comparison process. To make sure that the exact object will not be missed in the next frame, the search window should be big enough but more importantly, the area of searching should be close to the object we are looking for. As a result, different MTT algorithms apply different fluid motion models or motion estimation algorithms to predict and suggest the object possible location in the next frame [7, 24, 36]. Motion models can be for random movements (Brownian motion), directed movements or a combination of both [7, 36]. Next position of each object can also be predicted according to the movement history of the related track. Next position can be estimated by methods such as Kalman filter [7, 5].

As shown in Figure 1.2, after assigning objects to existing tracks and maintaining the tracks in inspection step, motion model or estimation model should be updated according to the new assignments. By this update, the location of objects in next frame can be predicted in the next step (predicting and gating). After that, objects of next frame will be retrieved. These five main steps (assigning objects, inspecting tracks, updating the models, predicting, and detecting objects in next frame) will run iteratively until there is no more frame available in the movie.

1.3.3 Multi-frame Tracking Algorithms

Instead of connecting objects of consecutive images to each other, multi-frame tracking methods look for the best possible tracks in the entire movie [43, 20, 30, 49, 5]. Frame-by-frame tracking algorithms have one main weakness; shape and color of objects might change during their movement. Therefore, there is a chance that by comparing the objects just between consecutive frames, two completely different objects get connected to each other. Since in most of the frame-by-frame tracking methods there is no history about features of those two objects, this connection might be accepted. As a result, in images with several similar objects frame-by-frame algorithms are more error prone than multi-frame tracking [39, 7].

Multi-frame tracking methods are also known as Multiple Hypothesis Tracking (MHT) algorithms [6, 5]. The main features of these methods are:

- They take all the possible connections between objects of different frames into consideration (by using gating methods the number of connections is reduced significantly)

- Possible tracks (which are also called as hypotheses in MHT) are sequences of these connections.

- For every hypothesis, track score is computed according to the similarity of objects in that hypothesis.

- Hypotheses with highest scores will be selected as final tracks.

Flow network is a Directed Acyclic Graph (DAG) that is recently used successfully in some multi-frame tracking algorithms [43, 39, 30, 49]. In this graph, every detected object is a node. For each edge that connects two nodes, capacity or weight of that edge is computed based on the dissimilarity between related nodes (i.e. objects). The direction of the edges is from nodes of previous frames to the nodes of the next frames. Nodes with a possibility of starting a track are connected to the birth node and nodes that can be the last presence of an object in the video are connected to sink node. If edges are considered as pipes, the ultimate goal is to find the K pipelines in the network with the least total cost of pipes to flow a fluid from birth node to the sink node. This goal is equivalent to finding the K-Shortest Paths (KSP) problem [30].

In Figure 1.3 an example of using Flow Network for MTT is shown. In this network, detected objects are shown as blue nodes. Object Oij is the jth detected object in the ith frame. Nodes are connected to each other by directed edges. Edges of this example just connect objects of consecutive frames to each other. Weights of edges are computed based on dissimilarity of related nodes (although not shown in the graph). Between objects of successive frames, not all objects are similar enough to be connected to each other (e.g. there is no similar enough object in frame number 1 to object $O24$). Objects which are good candidates for starting a track are connected to the birth node. Please note that there might not be a specific relation between frame number and connection to the birth node. For example in slurry images, objects are sinking and as a result, objects in the top part of the images are good candidates for starting a track. The same conditions are true for connecting the ending nodes to the sink node. After generating the graph, the goal is to find K paths from the birth node to the sink node which have the least total weight of the edges. Final tracks will be those paths without considering their connections to birth and sink node.

Despite their general better performance than frame-by-frame tracking al-

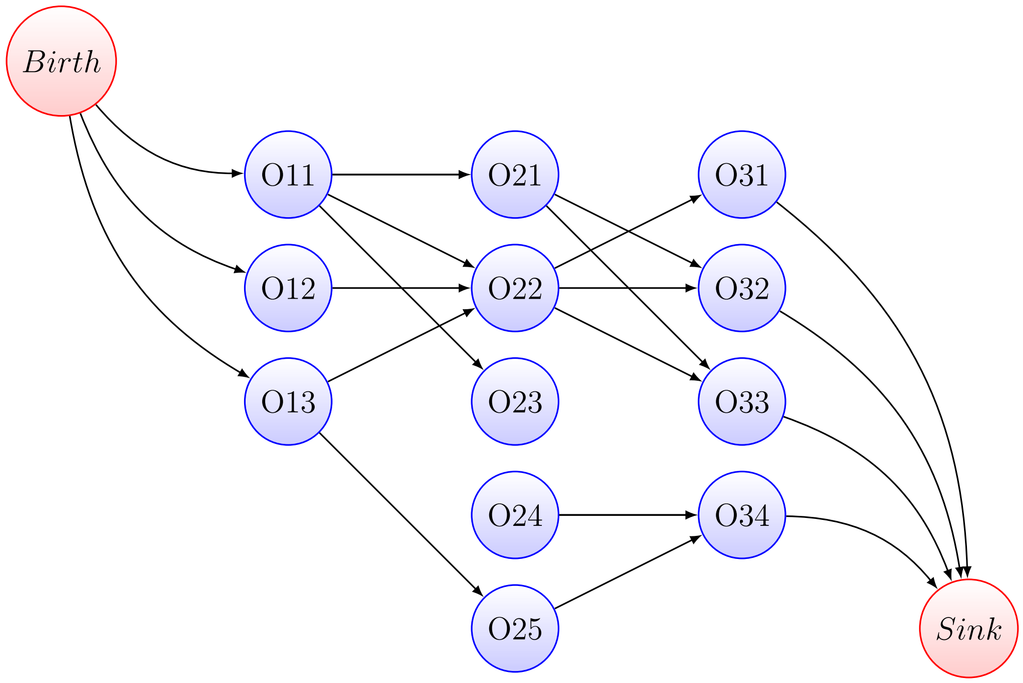


Figure 1.3: An example of using Flow Network for tracking multiple objects in sequences of images. Object $\mathbf{O}ij$ is the jth detected object in the ith frame. Connection between edges are directed and their weight is computed based on their dissimilarity of related nodes.

gorithms [7], multi-frame (i.e. MHT) methods are not good at running time and memory usage since every time a new frame with new objects is added, all of the possible tracks (hypothesis) should be updated. Some hypothesis may fail at the gating step and they should be removed. On the other hand, some other new hypotheses might get started. Updating all of the changes is very time-consuming. For keeping all the possible hypotheses, these methods also consume a huge amount of memory. These drawbacks are recently addressed in a greedy Flow Network based algorithm proposed in [30].

For speeding up the process of finding KSP in Flow Network and at the same time using less amount of memory, a greedy algorithm was proposed at [30]. This method suggests saving the least possible total cost for getting from the birth node to all detected objects in the current step (i.e. current frame). In the next frame, each detected object will be connected to an object in the previous frame with the following condition; previous frame object's total cost plus the connected edge cost to the current object should be the least comparing to other connections. For the next step, least possible costs should be updated just by adding the cost of the new connections. At the end (by getting to the sink node), objects (and their related path) with the least total cost will be selected.

By applying this greedy improvement there is no need to save all the network in the memory. Instead, only objects of the last frame need to know their path to the birth node and its cost. On the other hand, while the performance remains identical, this method reduces the run time from $O(N^3 log^2 N)$ to $O(KNlogN)$ where N is the total number of frames and K is the number of tracks [30].

1.3.4 Tracklet Stitching Algorithms

One of the main problems in cases that MTT algorithms are needed is that most of the times targets are moving in different directions and they also rotate. As a result, their appearance based features will change during the movie. Considering similar objects with similar shape and color around, this drawback can cause several misassigning between objects. On the other hand, although feature changing is inevitable, its effect is less in shorter time periods. For example, considering an object in two consecutive frames, its appearance is almost the same, but after several frames, it can be different noticeably. As a result, many algorithms are proposing tracklets (i.e. small tracks with the lifetime of fewer than 10 frames) in order to have more reliable features for targets during their movement [39, 48, 6, 30, 41].

Tracklets are in fact short sub-tracks of the main tracks. By using tracklets, instead of connecting objects to each other to form the tracks, similar tracklets will be connected and made the final track. This method is also known as tracklets stitching [6, 30]. Using tracklets provides features that will not change unpredictably in the upcoming frames such as average size of the object or its average color. In addition, if a misassigning happens between objects, this error only affects the rest of the tracklets that contain those two objects. Recently a novel Flow Network based method for tracklet stitching has been developed that uses tracklets as nodes in the Flow Network instead of each detected object separately [6].

In general, a very important factor to have good results is to have tracklets with good accuracy. There are different methods to generate the tracklets. For example in Singh2008 algorithm [41] whenever similarity between two consecutive objects is less than a threshold, tracklet will be finished. Shitrit2014 [39] first uses Flow Network with every object as a node in it. After computing the best tracks, this method cuts them to several smaller tracklets whenever two tracks get close to each other. The rational behind it is that misassignments always happen when objects are too close.

1.4 My Contribution to the Process of Computing Object Velocities

In order to compute velocity of floating objects, a system needs to have good performance in two major steps: **(1)** detecting the objects in each image and **(2)** tracking the objects in a sequence of images. We contributed to both steps. Chapter 2 is dedicated to the methods we developed and the experiments we performed for the first part of this project (i.e. detecting as many objects as possible in the least running time). Chapter 3 of this thesis is dedicated to the methods we implemented for tracking the objects. In the following sections, we bring a brief description of our contributions.

1.4.1 Comparing Existing Methods and Applying Tiling and Down-Sampling

Slurry images have some specific features and contain hundreds of objects with different sizes, in different shapes and different brightnesses (bitumen droplets are darker than sand particles). In addition, images suffer from poor illumination, and the background light changes in different parts of them. Here the goal is to detect most of the objects as fast as possible. In order to assess performance of previously developed methods on our datasets, we tested four global thresholding (MAX-ENT, MIN-ERR, Otsu and Iterative), seven local thresholding (2D-ENT, Niblack, Sauvola, Bernsen, Gatos, NICK, MAT) and five segmentation algorithms (GB, SLIC, SRG, M-SHIFT and SRM).

Image segmentation methods are time-consuming. In comparison, global thresholding algorithms are faster; however, they do not have good performances when there are light changing effects in the image. As our first attempt to improve the performance and decrease the running time, we proposed using two simple methods, image tiling and downsampling. More details about the results of applying existing methods on slurry images are provided in Section 2.3. Also, an extensive study about the effects of image tiling and downsampling on image segmentation and global thresholding methods can be found in Section 2.4.

1.4.2 Proposed Framework for Improving Image Thresholding Algorithms

We contributed to this area of research by proposing a framework that detects small objects using fast and accurate thresholding methods. Our framework iteratively detects relatively large and dark objects and then removes them from the image to give more chance to the relatively smaller or brighter objects to be detected. Any global or local thresholding algorithm can work with this framework properly. Although our proposed framework may use the global thresholding approaches, it does not have the main drawbacks of global thresholding methods (including being vulnerable to the background illumination changes). To evaluate our method, we compared it against ten different popular thresholding algorithms, including four global thresholding and six local adaptive algorithms. The main objects that we look for include sand particles and bitumen droplets which have relatively darker colors, while the image background is bright. Section 2.5 is dedicated to explaining this proposed framework in more details.

1.4.3 Tracking Objects by Applying Hungarian Assigning and Network-Flow To the Detected Objects

After detecting objects in each image, the next step was to accurately track the detected bitumen droplets and sand particles in a sequence of images. As mentioned earlier, there are two main categories of object tracking approaches: frame-by-frame approaches and multi-frame methods. In general, multi-frame tracking algorithms have better performance in other MTT applications. In comparison, frame-by-frame methods need less memory to run and can do all comparisons in one traverse of all images.

None of the existing methods for MTT has been previously evaluated and applied to the slurry images, and for the first time, we applied two tracking algorithms to track the detected objects: **(1)** a Hungarian assigning method (as a frame-by-frame based approach); and **(2)** K-shortest paths (KSP) based tracking in a Flow Network (as a multi-frame based approach). For the frame-by-frame tracking algorithm, objects of two consecutive images are assigned to each other by using the Hungarian method. For the Flow Network method (which is a multi-frame tracking algorithm), we found K-shortest paths in the network, where each detected object in each image was a node of the graph and the edge weights were computed by using the dissimilarity between the nodes. More details about the implementation and experiment results of these two methods are described in Section 3.3

1.4.4 Using Tracklets and How to Cut Them

The result obtained from applying tracking algorithms (Section 3.3) indicated that finding KSP in a Flow Network has a better performance than Hungarian assigning tracking algorithm. Therefore, we asked whether we can improve the performance of the Flow Network even further. As mentioned earlier, using tracklets instead of detected objects has been previously shown to improve the results in other applications. Therefore, likewise we were keen to see whether the same effects were observed in our specific application of the Flow Network in the slurry images.

There are several ways to cut the tracks into smaller parts to obtain the tracklets. We tested two of the existing methods. In addition, we proposed a two new cutting method to generate more reliable tracklets (which in turn results in generating accurate tracks at the end). We named the proposed cutting methods as Kalman filter cut and Hard cut. Hard cut divides a track into smaller tracklets with the same size. Kalman filter cut uses Kalman filter to cut a track whenever a big jump happens between the object's expected new location and its observed new location. More details of the methods and experimental results of using tracklets in the Flow Networks are explained in Section 3.4.

Chapter 2

Detecting Objects in Each Image

In this chapter, we describe the steps required to make sure objects are detected in each image as accurately as possible. It is preferable to detect bitumen droplets and track them in real time and as a result running time is also an important factor.

Some of the most famous algorithms of global Thresholding, local Thresholding and Segmentation are compared to each other with regard to our images in Section 2.3. To improve the accuracy and/or efficiency of the algorithms, two simple ideas can be; **(1)** tiling the image into smaller parts and apply the algorithm on each part separately, or **(2)** downsampling the images into smaller sizes with hope to have the same accuracy but in a less running time. In Section 2.4, we studied the effect of using tiling and downsampling methods on the performance of mentioned algorithms. At Section 2.5, we proposed a framework that is able to work with both global and local Thresholding algorithms and it will improve their performances. For evaluating any of the algorithms and methods and also the proposed framework, some ground truth images are required. We explained the process of making ground truth images in Section 2.1 before any other topics. Section 2.6 contains conclusion and summary.

2.1 Making Ground Truth Images

For comparing the thresholding methods, we needed to compare them with some ground truth images. We prepared our ground truth as binary images in which a user manually marks foreground pixels as one. The remaining pixels form the background and are labeled as zero (or vice versa). The size of slurry images is $2736 * 1500$ and they contain large quantities of bitumen droplets and sand particles. In general, original pictures can be categorized into two: **(1)** sparse, and **(2)** dense images. At the beginning moments of floating slurry in the flow tube, slurry images contain fewer objects (i.e. bitumen droplets and sand particles) and therefore slurry images are more transparent and sparse. However, after a while, the number of objects increases gradually and each image can contains hundreds of objects. We call the first group as sparse images and the latter group as dense images. In sparse images, there are fewer objects to detect and the image histogram is mostly unimodal to some extent. In comparison, in dense images, there are hundreds of bitumen droplets and sand particles in the hot water and images mostly have bimodal histograms. We manually labeled five sparse ground truths and five sparse ones by selecting all acceptable objects in an image with Adobe Photoshop and then replace them with zero; rest of the image is labeled as one. Each group of five images to be labeled are timely consecutive (five consecutive frames).

In making ground truth images, there are cases that some of the close objects are labeled as one big object. The reason is that the boundary between some adjacent bitumen droplets and sand particles could not be distinguished properly with bare eyes, therefore they were all labeled as one big object. Also, there are some scratches on the surface of the cylinder. These scratches are neither bitumen droplets nor sand particles, but without any knowledge about the shape of the objects, they are dark and big enough to be detected as an object in a two-dimensional image by a thresholding algorithm. We labeled bigger darker scratches as foreground and ignored the rest. On the other hand, in making ground truths, we selected as many objects as possible and there are some small or bright objects in ground truth images which are hard to detect by all of the thresholding methods.

2.2 Evaluating Detection Methods

We applied the existing algorithms and also proposed method on each ground truth image, and obtained a binary image. We then compared the binary result image against the manually labeled ground truth. We measured correctness score using Recall, Precision, and F-score. These three scores are computed pixel-wise; comparing the number of pixels detected as foreground by a thresholding algorithm with the number of pixels labeled as one in ground truth images. In addition, we used OCE-Score as a measure which is sensitive to under-segmentation and over-segmentation between detected objects of an algorithm with labeled objects in ground truth images. OCE-Score is an object based scoring method and gives a penalty for an inappropriately detected object. The output of the methods is a similarity score in the range of [0 1].

2.3 Comparing Thresholding and Segmentation Methods

To compare the the results of previous works with regard to our slurry images, we tested four global thresholding (MAX-ENT, MIN-ERR, Otsu and Iterative), seven local thresholding (2D-ENT, Niblack, Sauvola, Bernsen, Gatos, NICK, MAT) and five segmentation algorithms (GB, SLIC, SRG, M-SHIFT and SRM). These methods are described in Section 1.2.

Some of the algorithms have parameters which need to be tuned properly. Values we mention in the following are reached by testing different numbers to get the highest performance results. Global thresholding algorithms do not need any parameters to set. For 2D-ENT we used a sub-window with size 3*3. In MAT 10% of lower and 10% of the upper histogram are ignored and the sub-window size is 7*7. In Niblack K is -0.2, offset C is 10 and the sub-window size is 85*85. For Sauvola a bigger sub-window of 465*465 is used with $K = 0.34$ and R is equal to the maximum standard deviation in the image. To apply Bernsen algorithm sub-window size is 93*93 and if the intensity difference between the brightest and the darkest pixel in this sub-window is less than 5, we assume that area is homogeneous. For NICK K is -0.2 with sub-window 95*95.

Segmentation algorithms divide the image into smaller parts (i.e. smaller areas) with similar properties. In most of the cases, number of areas is more than two. It can be helpful as it can distinguish between hot water, sand particles, and bitumen droplets but first, there is a need to make sure segmentation algorithms can detect the objects properly. To do so, we applied some post processing on each of the algorithms in order to have binary images with detected objects in white and background in black. GB algorithm gives us several partitions. We think of the biggest partition as background and the rest as foreground. If checking for neighbors is based on the pixel distance, threshold K in the algorithm is equal to 2.5 and the minimum size of a segmentation cluster S is 50. For the case of finding neighbors based on Nearest Neighbor algorithm, threshold K is $\frac{2.5}{\sqrt{3}}$, the number of nearest neighbors N is 10 and the searching range R is 50. We tested SLIC with three different numbers of superpixels $K = \{500, 1000, 5000\}$. We then replace the intensity of each pixel by the average intensity of its super pixel. As the last post-processing step, we apply the MIN-ERR algorithm to find a proper threshold K. Pixels of every superpixel with an intensity average above K are parts of the foreground. In SRG, we select the brightest pixels as seeds and maximum intensity distance for breaking expanding loop is 12.5. To turn a Mean Shift algorithm's output to a binary image, we first convert it to a grayscale image and then apply the MIN-ERR thresholding method to find the objects. Spatial range to consider for moving is 40 and threshold for convergence is 9.

In order to increase the quality of the images and make circular objects easier to detect, we applied a Gaussian filter on each frame as a preprocessing step. In some of the cases, after running the algorithm, there are holes in the middle of the foregrounds. However, since sand particles and bitumen droplets are closed filled objects with no hole in them, we filled these holes as a post-processing step.

Table 2.1: Results of applying different thresholding and segmentation algorithms on sparse slurry images. The best results are for MAX-ENT and Nick methods.

	Spare Images - Average Running Times in Seconds				
Algorithms	**Recall**	**Precision**	**F_Measure**	**OCE_Score**	**Run Time (s)**
MAX-ENT	0.68	0.79	0.73	0.56	0.1
MIN-ERR	0.57	0.92	0.7	0.51	0.21
Otsu	0.96	0.01	0.02	0.004	0.16
Iterative	0.95	0.01	0.02	0.004	0.17
2D-ENT	0.67	0.79	0.73	0.54	13.22
Niblack	0.92	0.6	0.73	0.57	0.39
Sauvola	0.58	0.87	0.7	0.51	0.42
Bernsen	0.99	0.01	0.02	0.01	1.23
Gatos	0.65	0.81	0.72	0.54	1.48
Nick	0.63	0.9	0.74	0.56	0.4
MAT	0.78	0.04	0.08	0.02	493.95
GB-Grid	0.97	0.35	0.51	0.34	27.86
GB-NN	0.98	0.27	0.42	0.26	40.55
SLIC-500	0.89	0.23	0.36	0.18	3.68
SLIC-1000	0.69	0.68	0.68	0.42	5.65
SLIC-5000	0.51	0.94	0.66	0.48	22.14
SRG	0.97	0.01	0.02	0.002	2338.89
M-SHIFT	0.48	0.96	0.64	0.45	14463.62
SRM	0.59	0.84	0.69	0.52	41.19

The results of applying these methods on slurry images are shown in Table 2.1 and Table 2.2. Considering both sparse and dense images, the best algorithms are MAX-ENT and Nick thresholding algorithms with high accuracy rate and very fast running time. Segmentation algorithms are not as good as thresholding algorithms in detecting objects and moreover, they are significantly slower comparing to thresholding methods. Most of the algorithms have a better performance in dense images as the number of objects (including darker and bigger ones) are more and it makes them easier to detect.

2.4 Applying Tiling and downsampling

As mentioned earlier, although most of the global thresholding algorithms are super fast with reasonable good results, they are vulnerable against poor lighting in different parts of the image. If there is a way to select a smaller part of the image in which the illumination does not change significantly and apply

Table 2.2: Results of applying different thresholding and segmentation algorithms on dense slurry images. The best results are for MAX-ENT and Sauvola methods.

	Dense Images - Average Running Times in Seconds				
Algorithms	**Recall**	**Precision**	**F_Measure**	**OCE_Score**	**Run Time (s)**
MAX-ENT	0.8	0.9	0.85	0.68	0.11
MIN-ERR	0.74	0.94	0.83	0.65	0.22
Otsu	0.79	0.92	0.85	0.69	0.14
Iterative	0.78	0.92	0.84	0.68	0.16
2D-ENT	0.22	0.95	0.36	0.18	14.61
Niblack	0.84	0.91	0.87	0.73	0.41
Sauvola	0.85	0.91	0.88	0.74	0.44
Bernsen	0.95	0.77	0.85	0.64	0.87
Gatos	0.68	0.96	0.8	0.6	0.18
Nick	0.67	0.97	0.79	0.59	0.4
MAT	0.54	0.85	0.66	0.48	856.3
GB-Grid	0.98	0.71	0.82	0.62	28.1
GB-NN	0.97	0.76	0.85	0.67	40.53
SLIC-500	0.77	0.67	0.72	0.33	2.65
SLIC-1000	0.35	0.99	0.52	0.30	3.73
SLIC-5000	0.68	0.96	0.8	0.6	15.94
SRG	0.98	0.29	0.45	0.09	1025.44
M-SHIFT	0.81	0.91	0.86	0.7	14536.41
SRM	0.73	0.93	0.82	0.64	54.14

the same algorithm just on that smaller area, results can be improved. One simple idea is to divide the image into smaller parts, or in other words, tile the image with smaller sub-windows, and then apply the same thresholding algorithm on each tile separately. Figure 2.1 shows an example of applying Otsu method on the entire image, and also on four divided parts separately. In this example, we aimed for finding dark objects in the bright background. In the output, white areas are foreground while black means background. The third image is the result of dividing the image into four equal smaller images and apply Otsu on each of them. The image suffers from different background lighting in different parts of it and it is only by using tiling that we could detect both objects. As a drawback, running thresholding methods on several tiles will cost having longer running times (see Table 2.3 and Table 2.4).

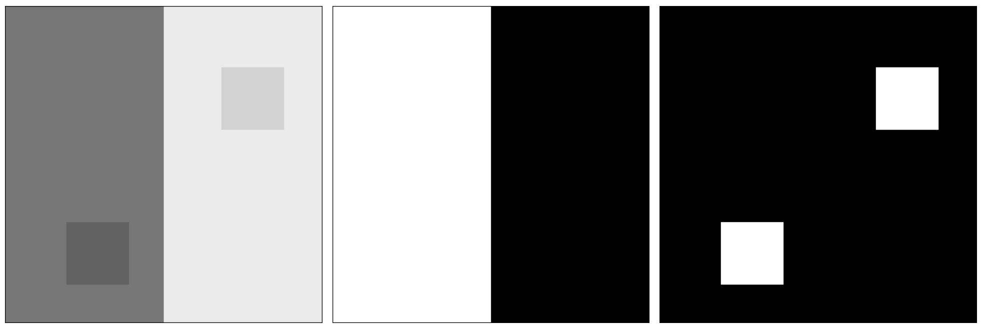

(a) (b) (c)

Figure 2.1: Advantage of using tiling for global thresholding algorithms. (a) original image, (b) result of applying Otsu algorithm on the entire image and (c) result of applying Otsu on four equally divided sub-images.

On the other hand, segmentation algorithms are designed for several purposes not just detecting background and foreground. They have different functionality, but most of them split the image into several similar segments. It can be helpful for separating background hot water, sand particles and bitumen droplets from each other. As also shown in Table 2.1 and Table 2.2, their well-known weakness is that most of them are time intensive.

One solution to address this problem is downsampling. For most of the segmentation algorithms, increasing the number of pixels running time would increase the running time. The run time can be reduced by applying the segmentation methods on downsampled images. On the other hand, the downsampled images will lose a noticeable number of pixels and cause the algorithm to have less information which is undesired for proper processing.

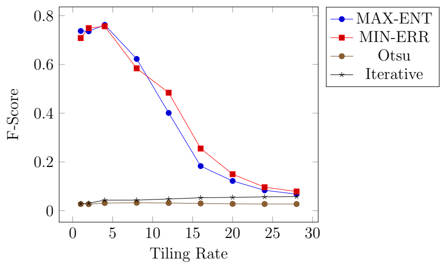
Tiling is also a helpful solution here. Instead of doing a time-consuming process, dividing the image into smaller sub-images and then segment sub-images one by one makes the whole process faster. Most of the segmentation algorithms need $\mathcal{O}(n^2)$ or more time to run, where n is the number of pixels [8, 3]. As a result, tiling works faster than applying the algorithm on the entire image.

The results of applying tiling on global thresholding and also segmentation algorithms, in addition to applying downsampling on segmentation methods, are reported in Section 2.4.1 and Section 2.4.2. These two ideas are not new [16, 25], but the experiments provide us helpful information. We benefit from them in our proposed framework for applying the right amount of dividing (more details in Section 2.5.2).

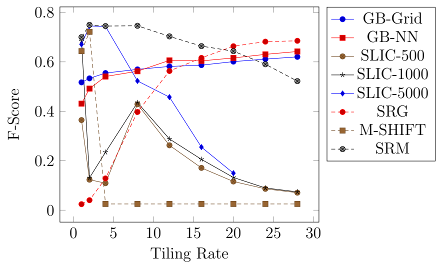
2.4.1 Tiling Experiments

In order to verify the effect of tiling on the performance of global thresholding and segmentation algorithms, we selected different tiling rates $t = \{1, 2, 4, 8, 12, 16, 20, 24\}$. In each experiment, the height of each image is divided into t pieces and the width of the image into $\frac{3}{2} * t$ as in our images width is longer than $\frac{3}{2}$ of the height. Figure 2.2a shows the average F-score of dividing sparse images into smaller parts within different division rates t and applying different global thresholding algorithms on them. When $t = 1$ we use the original image. The result of running the same experiments on segmentation methods is shown in Figure 2.2b. Average running time of the experiments on sparse images is presented in Table 2.3.

Dividing the image into smaller parts increases the run-time of global thresholding methods since a fast algorithm, which is $\mathcal{O}(1)$, will be run more than one time. Tiling also decreases the performance of global thresholding algorithms although it has a peak at earlier divisions. In segmentation algo-



(a)



(b)

Figure 2.2: F-Score vs Tiling Rates in sparse images, for (a) Global thresholding and (b) Segmentation Algorithms.

Table 2.3: Average running time of applying each method on tiled sparse images with different tiling rates t.

	Spare Images - Average Running Times in Seconds								
Algorithms	**t = 1**	**t = 2**	**t = 4**	**t = 8**	**t = 12**	**t = 16**	**t = 20**	**t = 24**	**t = 28**
MAX-ENT	0.11	0.27	0.18	0.31	0.50	0.70	0.94	1.22	1.53
MIN-ERR	0.19	0.21	0.22	0.24	0.26	0.28	0.31	0.35	0.39
Otsu	0.14	0.16	0.17	0.20	0.25	0.31	0.39	0.50	0.58
Iterative	0.20	0.16	0.17	0.21	0.27	0.32	0.39	0.49	0.62
GB_Grid	27.86	22.83	19.08	22.87	33.93	48.75	70.40	96.17	127.39
GB_NN	40.55	33.65	28.29	31.17	43.45	57.30	80.35	105.75	135.39
SLIC_500	3.68	3.99	4.59	5.77	6.87	7.25	9.47	15.52	15.30
SLIC_1000	5.65	6.94	7.91	9.08	14.82	19.18	16.71	33.98	34.11
SLIC_5000	22.14	28.46	32.98	45.50	62.89	171.29	144.06		
SRG	2338.90	795.04	439.49	220.80	145.97	115.85	101.16	91.77	92.86
M-SHIFT	14463.62	3912.48	704.41	288.87	210.56	176.59	154.84	142.67	133.15
SRM	41.19	8.06	6.43	5.28	5.55	6.10	6.91	7.81	8.85

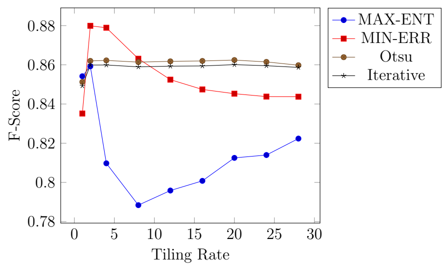
rithms, GB and the SRG are the only methods which have a better performance by tiling into smaller parts. Tiling has a good effect on SRG, M-SHIFT, and SRM and makes them faster. On the other hand, because of required steps for initializing and post-processing methods such as GB and SLIC, these segmentation algorithms work slower by increasing number of tiles. SLIC is dependent on the number of superpixels K, and as a result, by having more number of superpixels, running time would be longer.

The average F-score of dividing dense images into smaller parts within different tiling rates t and applying different global thresholding algorithms on them is shown in Figure 2.3a. When $d = 1$ we use the original image. The result of running the same experiments on segmentation methods is shown in Figure 2.3b. Average running time of the experiments on dense images is

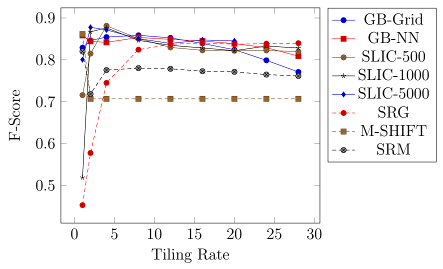
Table 2.4: Average running time of applying each method on tiled dense images with different tiling rates t.

	Dense Images - Average Running Times in Seconds								
Algorithms	**t = 1**	**t = 2**	**t = 4**	**t = 8**	**t = 12**	**t = 16**	**t = 20**	**t = 24**	**t = 28**
MAX-ENT	0.36	0.16	0.22	0.44	0.83	1.33	1.91	2.63	3.42
MIN-ERR	0.21	0.22	0.24	0.24	0.27	0.29	0.32	0.36	0.40
Otsu	0.13	0.14	0.14	0.16	0.19	0.23	0.28	0.35	0.43
Iterative	0.13	0.14	0.16	0.19	0.25	0.35	0.44	0.62	0.82
GB_Grid	28.10	23.08	20.17	23.22	34.70	48.88	70.17	96.11	127.28
GB_NN	40.53	33.28	29.21	32.13	47.67	56.72	80.49	104.97	136.14
SLIC_500	2.65	3.01	4.25	5.51	6.70	7.20	9.41	15.30	16.04
SLIC_1000	3.73	5.62	7.47	8.88	14.06	19.01	17.56	33.64	36.85
SLIC_5000	15.95	27.27	31.55	44.80	66.48	172.14	144.16		
SRG	1025.44	610.75	328.58	135.27	94.39	75.39	66.08	63.22	65.75
M-SHIFT	14536.42	3939.53	723.95	305.44	227.28	191.59	172.49	159.66	149.16
SRM	54.14	13.15	7.65	5.95	5.98	6.30	7.34	8.27	9.36

presented in Table 2.4.



(a)



(b)

Figure 2.3: F-Score vs Tiling Rates in dense images, for (a) Global thresholding and (b) Segmentation Algorithms.

Similar to sparse images, the same pattern of running time is observed for dense images. Algorithms have acceptable performance. It is because there are more dark objects in the images. Increasing tiling rate does not improve the performance. On the other hand, it is interesting that for the majority of these algorithms, tiling has no effect on their performance after a certain division rate which means it can make algorithms such as M-SHIFT, SRM, and SRG faster with the same performance. Comparing to segmentation methods, thresholding algorithms still have better performances on original image without any tiling and also they are still very faster.

2.4.2 Downsampling Experiments

To see the effect of downsampling on the performance, we tested different downsampling rates $d = \{1, 2, 4, 6, 8\}$. In each experiment, height and width of the image after downsampling (called as $newHeight$ and $newWidth$) are dependent on the downsampling rate d; $newHeight = \frac{height}{d}$ and $newWidth = \frac{width}{d}$ where $height$ and $width$ are the dimensions of the original image. In order to be comparable with ground truth images, we upsample the resulting image after segmenting the downsampled image to the original image dimensions. Both downsampling and upsampling are done with *resize* function of MATLAB. We do not apply downsampling on thresholding algorithms because they are fast enough. The average F-score of downsampling sparse and dense images within different division rates d and applying different segmentation algorithms on them can be found in Figure 2.4. $d = 1$ corresponds to the original image. Average running time of applying each segmentation algorithm on downsampled images with different downsampling rates is shown in Table 2.5 and Table 2.6.

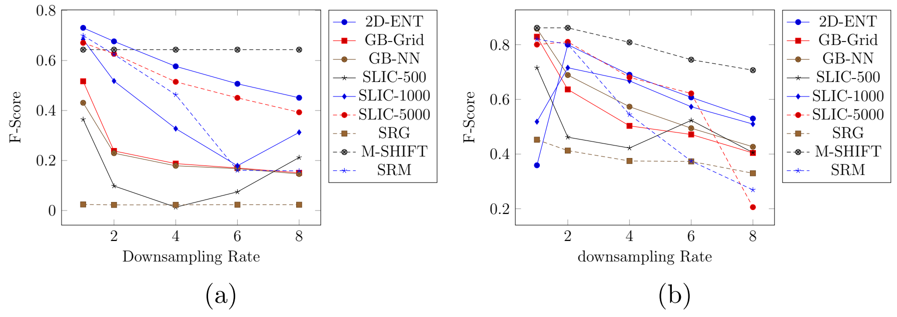


Figure 2.4: F-Score vs downsampling Rates in (a) sparse and (b) dense images for segmentation Algorithms.

Downsampling decreases the run time in all cases significantly. It reduces the performance of all segmentation algorithms except the M-SHIFT. In M-SHIFT case performance remains the same or decreases slightly.

In conclusion, since a real-time application is desired at the end, run time is a very crucial factor. Although downsampling and tiling had a good effect on increasing the speed of some segmentation algorithms, they are still much

Table 2.5: Average running time of applying each method on down-sampled sparse images with different downsampling rates d.

	Sparse Images Average Running Times in Seconds				
Algorithms	**d = 1**	**d = 2**	**d = 4**	**d = 6**	**d = 8**
GB_Grid	27.86	5.97	1.30	0.61	0.40
GB_NN	40.55	8.63	1.92	0.89	0.59
SLIC_500	3.68	1.14	0.43	0.31	0.25
SLIC_1000	5.65	1.69	0.57	0.35	0.29
SLIC_5000	22.14	6.44	1.88	0.95	0.62
SRG	2338.90	137.65	3.77	3.63	3.87
M-SHIFT	14463.62	801.20	37.17	9.11	3.86
SRM	41.19	9.43	3.27	2.83	1.81

Table 2.6: Average running time of applying each method on down-sampled dense images with different downsampling rates d.

	Dense Images Average Running Times in Seconds				
Algorithms	**d = 1**	**d = 2**	**d = 4**	**d = 6**	**d = 8**
GB_Grid	28.10	6.04	1.49	0.67	0.42
GB_NN	40.53	8.65	2.02	0.96	0.65
SLIC_500	2.65	1.14	0.42	0.33	0.33
SLIC_1000	3.73	1.13	0.45	0.30	0.26
SLIC_5000	15.95	4.26	1.33	0.78	0.55
SRG	1025.44	110.50	7.94	3.36	1.53
M-SHIFT	14536.42	798.45	38.96	9.43	4.10
SRM	54.14	8.70	2.65	2.22	2.05

slower than thresholding methods with less accuracy (worse performance). Tiling increases the performance in earlier division rates for MAX-ENT and MIN-ERR, but it had a bad effect on thresholding algorithms in general. Therefore, using tiling for improving the performance is not recommended.

2.5 Recursive Iterative Thresholding Framework

One of the problems with thresholding algorithms especially in cases such as ours is that dark objects (i.e. sand particles and/or bitumen droplets) shadow bright objects. In global thresholding algorithms, dark objects make the threshold value to be something small while these brighter objects normally have a brightness very close to the thresholding value. In local thresholding, shadow problem may happen because of being affected by a dark object in the neighborhood. This dark object decreases intensity average and increases standard deviation of the sub-window. As in most of the cases, standard deviation has a negative weight on computing the threshold value for the current pixel, having a dark object around means having a less threshold value and missing brighter objects. If a small window (for example b=3) is used, the probability of detecting noises as foreground goes higher and by increasing the size of the window. smaller or brighter objects would be missed.

In this section, we present a framework for detecting all objects including the dark ones while retrieving brighter and smaller objects without mislabeling noise as objects. This method is based on some conditions about our project and images we have, such as:

- There are hundreds of objects in each image and a noticeable number of them are small. They may have intensities similar to that of the background.
- Bigger objects are normally bitumen droplets that are also darker, and therefore, easier to be detected. In comparison, the smaller brighter objects are mostly sand particles.

2.5.1 Iterative Thresholding Framework for All Thresholding Algorithms

In the proposed method, we iteratively apply a thresholding algorithm on the original image, select the biggest detected objects among all recognized objects and then remove them from the image. We repeat these steps until every single detectable object is detected in this level of processing. Since big objects are usually darker ones, by removing them from the image, thresholding algorithms should focus on a smaller range of intensities to select the proper threshold value. As a result, the chance of detecting brighter objects in the next iteration is higher.

Removing the biggest detected objects from the image and preparing it for the next iteration is dependent on the thresholding algorithm that is used. For global thresholding algorithms that do their processes on intensity values of the image, we change the color of candidate objects to black with an intensity value equal to zero. We set that the first iteration is on intensity values in the range of [0,255], but for the next iterations, acceptable intensity values are in the range [1,255]. For local thresholding algorithms, we replace the best candidate pixels with the estimated background or an average of background pixels in a neighborhood.

We potentially could set the number of biggest candidates to be selected in each iteration (n), as a constant number, or a number increasing per iteration with a constant growing rate. However, we decided to set n as an increasing number that increases in each iteration with a growing rate that increases in each iteration. The reason is that there are a huge number of small similar objects in every image with almost the same size and same intensity. Bigger and darker objects are more distinguishable according to their area and intensity. However, small objects are very similar. Since small objects have the similar features and there are hundreds of them in each image, it is better to detect and remove them as much as possible in every iteration. By selecting this approach, the iterative method terminates more rapidly with the same performance. The algorithm will terminate if it cannot detect any more objects in two consecutive iterations; in other words, n does not increase for the last iteration. Another termination condition is when the size of the biggest detected object exceeds a constant value K. In the latest case, the detected object will be a part of the background, and since having any objects that big is not correct, the algorithm will not accept that iteration. The flowchart of the proposed framework is shown in Figure 2.5.

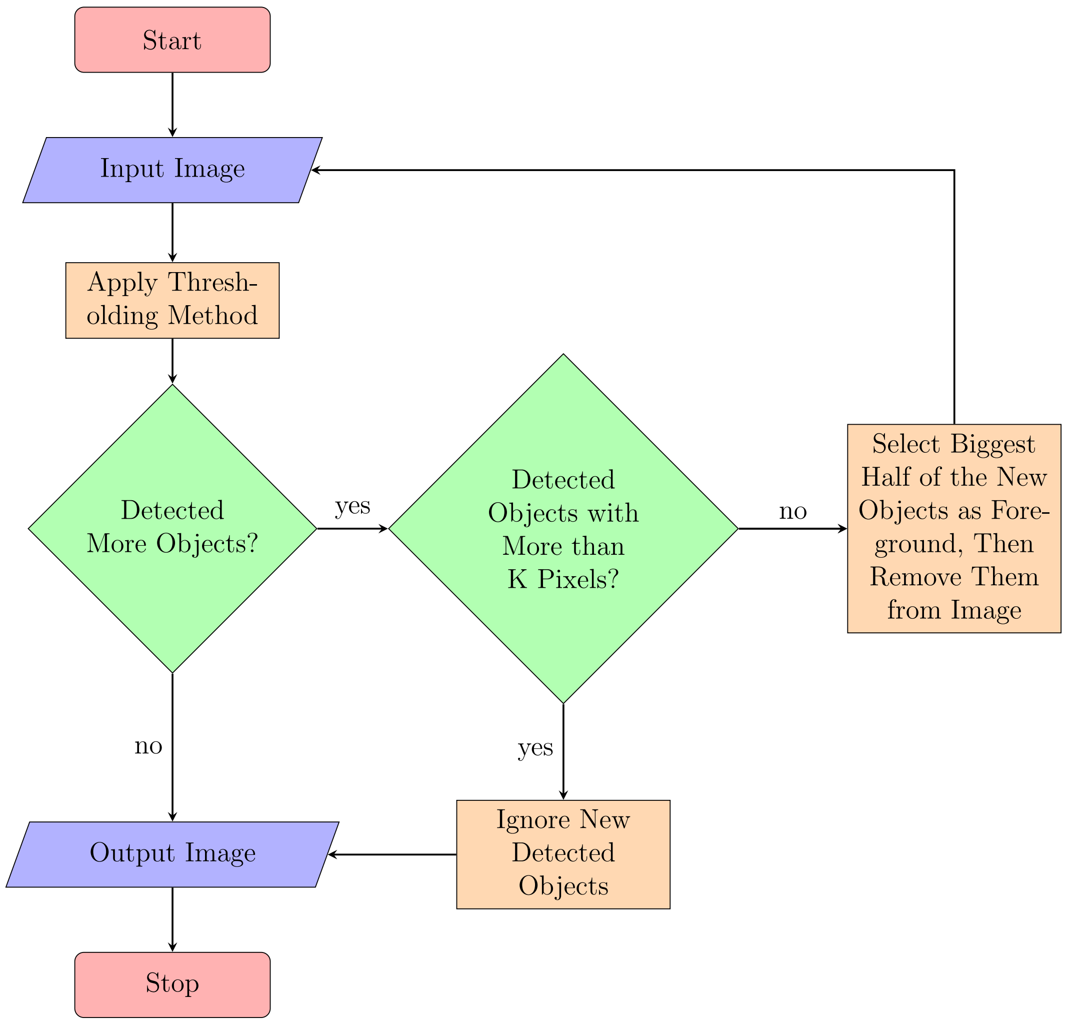

Start
Input Image
Apply Thresholding Method
Detected More Objects?
yes
Detected Objects with More than K Pixels?
no
Select Biggest Half of the New Objects as Foreground, Then Remove Them from Image
no
yes
Output Image
Ignore New Detected Objects
Stop

Figure 2.5: Flowchart of Proposed Iterative Framework

According to the result reported from Section 2.3, MAX-ENT has one of the best performances of a global thresholding method and it is significantly fast. NICK is also the best local thresholding method for our goal. Therefore, we decided to use these two thresholding methods to test in our proposed framework.

2.5.2 Recursive Iterative Framework for Global Thresholding Algorithms

As mentioned earlier in Section 1.2.3 and Section 2.4, the main weakness of global thresholding algorithms is their bad performance in poor lighting images. When we use a global thresholding method in our framework, the proposed iterative method is also weak against poor illumination which is a common situation in our images. To address this problem, first we apply the iterative procedure on the original image. Result image contains detected objects in black and the rest of the pixels have the same intensity as before. Then the resulting image will be divided into four sub-images and recursively the same method will be applied on each sub-image until some termination conditions are met. The last result will be achieved by running a binary OR operation among images of all these levels. By making the image smaller in each level of recursive processing and also eliminating big darker objects, the chance of detecting smaller objects will be higher and it is more robust against poor lighting in the image.

There are two termination conditions here: **(1)** size of the sub-image should not be less than $w*h$ and **(2)** the intensity difference between the darkest and the brightest pixels in that sub-image should be more than a constant value m. Selecting these three parameters can be manual or automatic based on image features. Based on trying different values on our images and comparing the results with ground truth images, we realized the best performance happens when $w = 400$, $h = 400$, $m = 10$. Going deeper in the recursive process causes the algorithm to detect noises as foreground objects. As the process does not go deeper, the program never met the intensity difference m termination condition, but by selecting higher values for m, the objects with intensities very similar to the background would be missed.

Table 2.7: Results of using proposed framework with the best global and local thresholding methods in sparse images.

	Sparse Images				
Algorithms	**Recall**	**Precision**	**F_Measure**	**OCE_Score**	**Run Time (s)**
MAX-ENT	0.68	0.79	0.73	0.56	0.10
Proposed Method Recursive Iterative (via MAX-ENT)	0.78	0.71	0.74	0.59	2.91
NICK	0.63	0.90	0.74	0.56	0.40
Proposed Method Iterative (via NICK)	0.64	0.89	0.74	0.56	3.47

Table 2.8: Results of using proposed framework with the best global and local thresholding methods in sparse images

	Dense Images				
Algorithms	**Recall**	**Precision**	**F_Measure**	**OCE_Score**	**Run Time (s)**
MAX-ENT	0.80	0.90	0.85	0.68	0.11
Proposed Method Recursive Iterative (via MAX-ENT)	0.93	0.80	0.86	0.64	8.29
NICK	0.67	0.97	0.79	0.59	0.40
Proposed Method Iterative (via NICK)	0.77	0.95	0.86	0.70	5.26

2.5.3 Recursive Iterative Thresholding Framework Implementation

To assess the performance of our proposed method, we applied our framework to the same set of ground truth images. Our framework is general and can be used with any global and local thresholding algorithms. We selected the global and local thresholding methods with the best performances to be used in our framework. The global thresholding algorithm with the best performance is MAX-ENT and the best performance among local thresholding algorithm was achieved by NICK (See Table 2.1 and Table 2.2). For testing MAX-ENT in our framework and based on trying different levels of dividing the image in the recursive process, we realized it is better to stop dividing the image into smaller parts after two levels. We found that if we keep dividing the image into smaller parts it may detect the entire sub-image as a homogeneous foreground area. When using NICK as a local thresholding algorithm in our framework, pixels of detected objects in the image will be replaced with the average intensity of their neighborhood. The same preprocess (applying Gaussian filter) and post-process (filling the holes) are also applied here. Results are shown in Table 2.7 and Table 2.8.

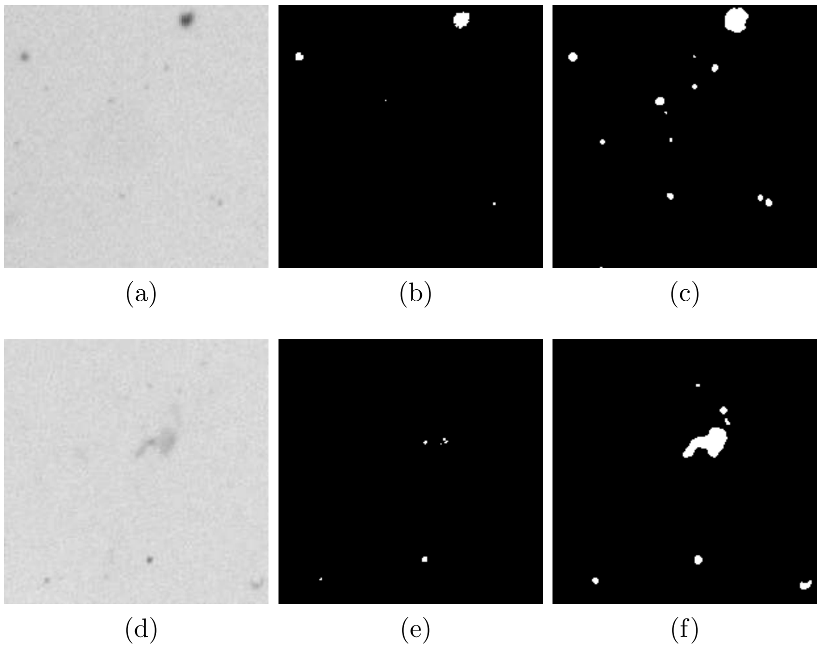


Figure 2.6: Examples of object detection with proposed framework via MAX-ENT thresholding algorithm. (a),(d) Examples of original image, (b),(e) objects detected with MAX-ENT method and (c),(f) objects detected by using MAX-ENT in the proposed framework.

The results show that our proposed method has a better performance than the best methods from the previous section. The main improvement achieved by using our proposed framework when compared to the other previously published methods is in its ability to detect smaller and brighter objects more accurately and more often. The small and bright objects are not easy to detect as they are more similar to the background, and also they are considered as noise in the other methods. Detecting more objects means achieving higher recall rates and in all cases, proposed framework increases recall of thresholding algorithms. It is also noticeable that since the goal is about detecting more small objects, in an image with hundreds of objects and while recall is a pixel-wise scoring method, even one percent increase in recall rate means having more than ten smaller objects. On the other hand, because of running several iterations and more specifically, processing an image in different levels of recursion in global thresholding algorithms, output images of the framework may suffer from a slightly under-segmentation problem and some close objects

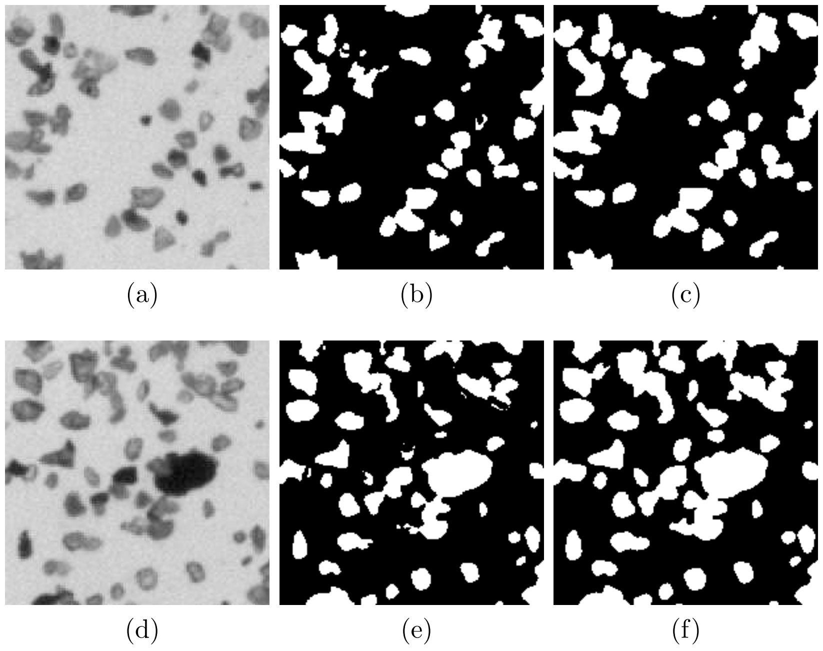


Figure 2.7: Examples of object detection with proposed framework via NICK thresholding algorithm. (a),(d) Examples of original image, (b),(e) objects detected with NICK method and (c),(f) objects detected by using NICK in the proposed framework.

are detected as one big object. It leads to lower precision rates. In contrast, in all cases, F-score as a combined measurement is improved by our method. Because of running multiple iterations, the run times of our framework are longer. Unlike precision and recall which are pixel-wise measurements, OCE is an object based scoring method and according to experiment results, the OCE is improved by our framework, as it is able to detect more objects correctly.

NICK thresholding algorithm has higher recall rates with our framework especially in dense images, but its performance is not changed that much in sparse images. The reason is that in the sparse images, objects are far from each other and the probability of being affected by a dark object in the neighborhood is less. As a result, if an object is not detected as an object and there is no dark object around, it will not detect as foreground in next iterations either. This condition happens rarely in dense images.

Examples of detecting small and bright objects by the proposed method and comparing them with results of MAX-ENT and NICK are shown in Figure 2.6 and Figure 2.7 respectively. Objects detected by our method have bigger areas than those of thresholding algorithms. The reason is that around each object there is a layer of pixels, which are not part of the foreground and also these pixels are not as bright as background pixels. This makes our algorithm detect them as part of the object.

2.6 Conclusion

In this chapter, we presented an iterative framework that is suitable for detecting smaller and brighter objects in an image accurately. This framework is compatible with both global and local thresholding methods. In the case of global thresholding methods, it is recommended to apply a recursive dividing approach to relax the effect of poor illumination on detection processes. In order to evaluate the performance of our method, we compared it to the well-known global and local thresholding algorithms. We first compared performances of four famous global thresholding, seven local thresholding algorithms, and five segmentation algorithms together on our ground truth images. We selected the top global and local thresholding algorithms by considering their F-measure, OCE score and run time. Other experiments such as tiling the images or downsampling them were also tested, but the results were not satisfactory. We then run our proposed method using these top global and local thresholding algorithms and compared the results against the best candidates of each group. All comparisons are done by using slurry images and the purpose is to detect bitumen droplets and sand particles as many and as accurately as possible. Results show that our method is able to detect more objects including brighter and smaller ones and it also performs better, but slower than the best algorithms of each group.

Chapter 3

Tracking Objects in Sequences of Images

By comparing some of the famous thresholding and segmentation algorithms, it is realized that MAX-ENT and NICK have the best performances for running on slurry images. Evaluating proposed Recursive Iterative Framework indicates that this framework can work better but it needs noticeably more running time. As a result, for the further studies, we just worked with the results of the MAX-ENT global thresholding algorithm.

After detecting objects properly in each image, the second main step is to track the detected objects in sequences of slurry images. By tracking any specific object in a video, a list of positions in different images will be made at the end. A fast algorithm which can track objects correctly is a suitable one because the aim is to track particles in real time.

Objects of slurry images do not blink or disappear temporarily between their movement (a case that happens in blood cell imaging [36, 9]). Therefore, an acceptable tracking method connects good candidates for several frames without missing any frames in between. A track is finished whenever the tracked object is out of the image boundary. In addition, objects can collide with each other in a few situations. In this study, we have developed a method to track objects in slurry images, and to the best of our knowledge, this is the first study that applies a method for tracking objects in slurry images. The main focus of our project was to test and evaluate the performance of different applicable methods. Handling cases such as merging or splitting of the objects is out of the scope of this study and is left for future work.

There are hundreds of detected objects in each image. Tracking all of these objects simultaneously would be very difficult, if not impossible. As a result, we focused on tracking the most important floating objects which are bitumen droplets. Bitumen droplets are darker objects comparing to sand particles. Therefore a good way to separate them out is by selecting darker objects. Since we used the MAX-ENT to detect objects, all the objects should have a brightness less than a global threshold T. Selected darker objects are the ones which had a brightness less than $T/2$. Tracking brighter objects is a part of the future works.

In this chapter, we describe implementation and evaluation of different tracking methods. For evaluation of the methods, some ground truth tracks are needed. Section 3.1 explains the required steps for generating a Graphical User Interface (GUI) to help a user generate ground truth tracks faster and easier. The evaluation process is described in Section 3.2. In Section 3.3 we mention the implementation of two tracking methods; a frame-by-frame based algorithm and a multi-frame based method. These two methods are then evaluated according to ground truth tracks. Section 3.4 describes the methods of improving the tracking performance by using tracklets in Flow Network. Section 3.6 contains conclusion of this chapter.

3.1 Designing a Ground Truth Generator GUI

We designed a GUI using MATLAB to help a user generate ground truth tracks in a very short time. By running the GUI, the beginning frame will be shown to the user. If T is the global threshold value computed by the MAX-ENT algorithm to separate foreground from background, GUI shows darker detected objects which have an intensity less than $T/2$ with a yellow square around them. User can select any of those yellow squares as the starting position of a certain object. GUI then shows the next frame with the same condition and user should select the right yellow square that contains the same object in that frame. This procedure continues until user realizes that the specific object is out of the image boundaries and it cannot be tracked anymore. Therefore the user stops that round and the track will be saved in the memory. The user can finish the program or continue to track another object. If the user decides to continue, GUI starts over and shows the first frame again.

There are many objects that appear later and the user cannot select them from the beginning frame. By using proposed GUI, the user can skip the frames and move on to the frame in which an object appears for the first time. If user realizes there was a mistake in his object selection in previous frames, he can easily go back in the frames and reselect another object instead.

To help user selecting a right object, GUI marks all the previously selected objects for existing tracks with a blue plus sign on them. Figure 3.1 shows an image of the proposed GUI. In this example, two tracks are already selected and to select a new object for initializing a new track, user has skipped 8 first frames and moved forward to the ninth frame. In this frame a new object appeared at the top of the image and user wants to select it by pointing the mouse at it (intersection of vertical and horizontal black lines).

By using the proposed GUI, we made 41 ground truth tracks for further evaluations. These tracks contain mostly big objects which were easy to track. They can start anywhere in the image and end anywhere else. As a knowledge about our images, we know that bitumen droplets are sinking and therefore we are interested in tracks which begin at the top part of an image and end at the bottom part of another image. We also know that height of each image is 1500 pixels. We considered objects with a distance less than 500 pixels from the top of the image as good candidates to start a track. Similarly, we assumed objects with a distance less than 500 pixels from the bottom of the image are good candidates to finish a track. After applying these constraints, 29/40 tracks remained. These formed our ground truth tracks that we used to evaluate tracking algorithms. Moreover, after having produced tracks by an algorithm, only those tracks that are descending in the images and have the same beginning and ending conditions are considered as final tracks.

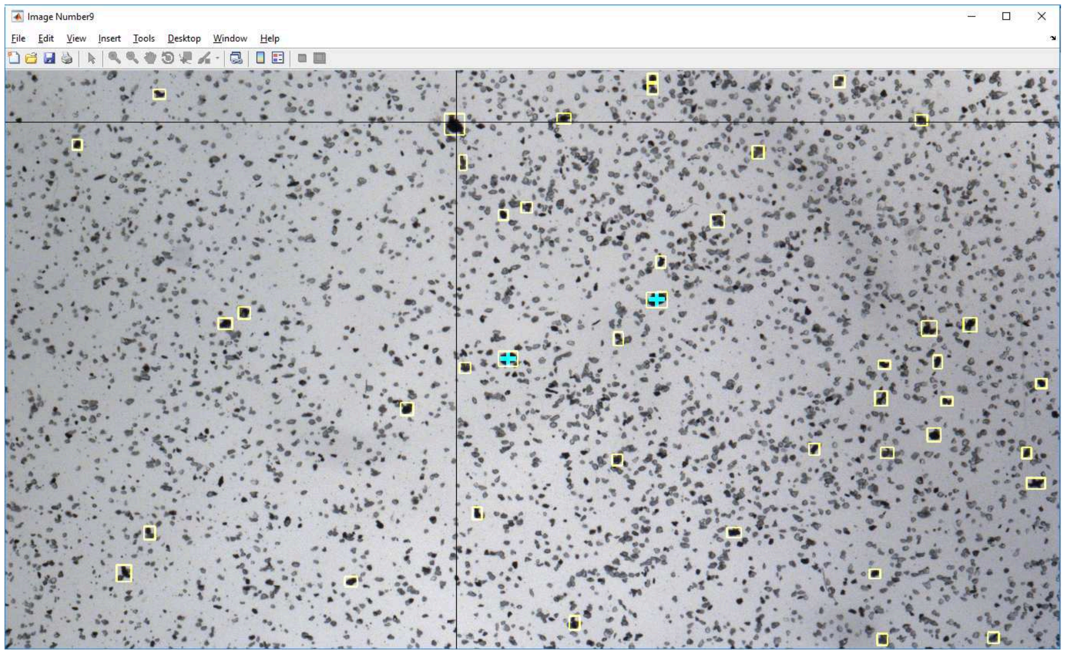


Figure 3.1: An example of running proposed GUI for generating ground truth tracks. Bitumen droplets and sand particles which were dark enough to detect as candidate objects are shown by yellow squares. To make sure the user does not select any part of the other tracks by mistake, already selected objects are marked with blue signs. In this frame a new bitumen droplet has appeared at the top of the image and user aims to select it (the one at the intersection of vertical and horizontal black lines).

3.2 Evaluating Tracking Methods

In order to evaluate the performance of each tracking algorithm, we compared the results of each algorithm against the ground truth tracks using the two following approaches; **(1)** Between Frames Based Evaluation and **(2)** Track Based Evaluation. In the following sections, we describe these two tracking evaluation methods.

3.2.1 Between Frames Based Evaluation (BFB)

Each tracking algorithm generates several tracks. Between any two consecutive frames, each track has either zero or one certain connection. Since a tracking algorithm generates several tracks, there are multiple connections between any two consecutive frames. The same condition holds for the ground truth tracks. The precision and recall between two successive frames (e.g. ith and $(i+1)$th frames) can be computed as:

$$Precision(i) = \frac{NTC(i)}{NC(i)} \tag{3.1}$$

$$Recall(i) = \frac{NTC(i)}{NGT(i)} \tag{3.2}$$

where $NTC(i)$ and $NC(i)$ are respectively the number of true connections and the total number of connections between ith and $(i+1)$th frames generated by the tracking algorithm. $NGT(i)$ is the total number of ground truth connections between the ith and $(i+1)$th frame. The average between frames based precision and recall for the entire movie are computed as:

$$Precision\_BFB = \frac{\sum_{i=1}^{N} Precision(i)}{N} \tag{3.3}$$

$$Recall\_BFB = \frac{\sum_{i=1}^{N} Recall(i)}{N} \tag{3.4}$$

where N is the number of frames in the video and BFB stands for Between Frames Based evaluation. $Recall\_BFB$ is similar to detection rate used in other multiple target tracking (MTT) papers [30, 46, 49] for evaluating and comparing tracking methods. F-Score of BFB evaluation is computed as:

$$F\_Score\_BFB = 2 * \frac{Precision\_BFB * Recall\_BFB}{Precision\_BFB + Recall\_BFB} \tag{3.5}$$

3.2.2 Track Based Evaluation (TB)

The second evaluation method considers the generated tracks as a whole, and measures Precision and Recall as follows:

$$Precision\_TB = \frac{NTT}{NT} \tag{3.6}$$

$$Recall\_TB = \frac{NTT}{NGTT} \tag{3.7}$$

where NTT is the number of true tracks, NT is the total number of tracks created by the algorithm and NGTT is the number of ground truth tracks (which is equal to 29 in our study). Here TB stands for Track Based evaluation.

In order to measure the number of true tracks (NTT), we first computed the accuracy $A(i)$ for each track i as the number of times the track i connections match the ground truth connections divided by the length of the track i. Then we used an accuracy threshold of t between 0 and 1, and identified all the tracks with the accuracy $A(i)$ greater than or equal to the threshold t. These tracks formed the set of true tracks. In our experiments, we used four different accuracy thresholds for t $(40\%, 60\%, 80\%, 100\%)$. The tracks with $A(i) = 1$ perfectly match to the ground truth tracks.

The drawback of this evaluation is that the lengths of the accepted tracks (i.e. $A(i) \geq t$) are not taken into consideration. Therefore, two tracks one with a length of 3 frames and another with a length of 30 frames may have the same accuracy $A(i)$ value. This is not appropriate for performance evaluations, as there is more interest in finding the tracks with relatively higher lengths that match to the ground truth tracks. To take this point into consideration, we used the following formula.

$$Weighted\_Precision\_TB = \frac{\sum_{i=1}^{n} L(i) * A(i)}{\sum_{i=1}^{NT} L(i)} \tag{3.8}$$

$$Weighted\_Recall\_TB = \frac{\sum_{i=1}^{n} L(i) * A(i)}{\sum_{j=1}^{NGTT} L\_GT(j)} \tag{3.9}$$

L(i) is the length of ith detected track and L_GT(j) is the length of jth ground truth track. The F-Score of TB evaluation is computed as:

$$Weighted\_F\_Score\_TB = 2 * \frac{Weighted\_Precision\_TB * Weighted\_Recall\_TB}{Weighted\_Precision\_TB + Weighted\_Recall\_TB} \tag{3.10}$$

For TB evaluation we tested four different accuracy rates $A(i) = \{40\%,$

60%, 80%, 100%}. Tracks with accuracy less than 40% are the tracks in which 60% of their lifetime they do not match with any ground truth tracks. Since we are not interested in these tracks we did not use accuracy rates of less than 40% in our evaluations.

3.3 Applying Frame-by-frame and Multi-frame Tracking Algorithms to Slurry Images

Up to our knowledge, this is the first attempt of applying tracking algorithms to the sequences of slurry images. As the first step, we evaluated one method of each group of tracking algorithms (frame-by-frame and multi-frame tracking algorithms) on our data sets. The performance of each algorithm is evaluated by TB and BFB evaluation methods by using the ground truth tracks.

3.3.1 A Frame-by-frame Tracking Algorithm

As a frame-by-frame tracking algorithm, we used Hungarian assigning method [26, 41] to find the best assignments between detected objects of any two consecutive frames. In general, Hungarian assigning algorithm finds connections between members of two groups with the least cost calculated based on a dissimilarity matrix. In our tracking method, each row of the dissimilarity matrix corresponds to one object of the current frame, each column corresponds to objects of the previous frame, and each element value corresponds to the dissimilarity between the corresponding objects from the two consecutive frames. By using Hungarian method, any object in the current frame will be connected to at most one object of the previous frame. Similarly, objects of previous frames can be connected to at most one object in the current frame.

In order to obtain a dissimilarity matrix, we computed the similarity between pairs of objects from two consecutive frames using the following features: **(1)** Number of Pixels (Shape); **(2)** Brightness (Color); and **(3)** X and Y coordinates (Position). We computed the similarity between pairs of objects according to any of these features using Equations 3.11 to 3.14, where $P1$ is a detected object in the previous frame and $P2$ is an object in the current frame. In these equations, $NP(P)$ is the number of pixels in object P, and $I(P)$ is the average intensity of pixels in object P. $X(P)$ and $Y(P)$ are X and Y coordinates of the center of object P. Parameters X_Diff_Max and Y_Diff_Max are described in the next paragraph. All three similarity measurements, $Shape\_Sim$, $Color\_Sim$ and $Position\_Sim$, get values between 0 and 1, where zero means no similarity and 1 means exactly the same.

$$Shape\_Sim(P1, P2) = min(\frac{NP(P2)}{NP(P1)}, \frac{NP(P1)}{NP(P2)}) \tag{3.11}$$

$$Color\_Sim(P1, P2) = min(\frac{I(P2)}{I(P1)}, \frac{I(P1)}{I(P2)}) \tag{3.12}$$

$$X1 = X(P1), X2 = X(P2), Y1 = Y(P1), Y2 = Y(P2) \tag{3.13}$$

$$Position\_Sim(P1, P2) = 1 - \frac{\sqrt{(X2 - X1)^2 + \frac{(Y2-Y1)^2}{4}}}{\sqrt{(X\_Diff\_MAX)^2 + \frac{(Y\_Diff\_MAX)^2}{4}}} \tag{3.14}$$

In our images, the objects mostly sink (move downwards), and in fewer cases, they float up (move upwards). In general, the objects vertical movements are more than their horizontal ones. Therefore, in Equation 3.14, the distance between vertical positions of two objects is divided by 4 to relax its effect on the equation. In general, if two objects in consecutive images have any of the following conditions, we consider them as different objects.

$$\begin{aligned} |X2 - X1| &> X\_Diff\_MAX \\ Y2 - Y1 &> Y\_Diff\_MAX \\ Y1 - Y2 &> Y\_Diff\_MIN \end{aligned} \tag{3.15}$$

The first condition indicates that objects can move horizontally in both left and right directions, but their horizontal movement is limited by X_Diff_MAX pixels per frame. The second condition indicates that the objects sink, but their downwards movement is not more than Y_Diff_MAX pixels between consecutive frames. Last equation ($Y1-Y2 > Y\_Diff\_MIN$) considers floating up movements which can happen on few occasions, and it indicates that the upwards movements are limited by Y_Diff_Min pixels between consecutive frames. Based on the knowledge about our images, such as the size of the images and approximate velocity and movement of objects, we set those three parameters as follows:

$X\_Diff\_MAX = 100, Y\_Diff\_MAX = 400, X\_Diff\_MIN = 50.$

We computed the similarity between pairs of objects ($P1$ from the previous frame and $P2$ from current frame) only if they were located in the horizontally and vertically constrained distances defined above.

By considering three features (i.e. shape, color, and position) similarities, we computed the overall similarity between objects $P1$ and $P2$ as follows:

$$\begin{aligned} Similarity(P1, P2) &= \alpha * Shape\_Sim(P1, P2) \\ &+ \ \beta * Color\_Sim(P1, P2) \\ &+ \ \gamma * Position\_Sim(P1, P2) \end{aligned} \tag{3.16}$$

where α, β and γ are the weights assigned to shape, color and position features respectively, and indicate the relative importance of each similarity feature. In slurry images, the feature that makes the objects distinguishable the most is the shape of objects. Especially big objects (which are bitumen droplets most of the time) have different shapes. Since the focus in this project is on tracking bitumen droplets, we assigned the highest similarity weight to the shapes of the objects. Intensity or color is another object feature that is stable most of the time as the object moves between frames, and therefore it can be used to track the object in different frames. However, bitumen droplets have similar intensities. Therefore, intensity feature should have less weight compared to the shape feature. In general, objects move downwards in a sequence of images but their exact movement is unknown to us. We already set a boundary for the acceptable range of movements in the horizontal and vertical directions. However, in these boundaries, different objects have similar chances of getting connected (i.e. false connection rate can be high) if the position feature weight has a very high value. Therefore, the position similarity weight (γ) should get the smallest value among the three features weights. By considering these constraints and trying different values, the following weights led to the best performance during the experiments: $\alpha = 0.6$, $\beta = 0.3$, $\gamma = 0.1$.

At the end, we used the similarity between two objects (a value that is in the range of zero to one) to compute the dissimilarity value between pairs of $P1$ and $P2$ objects:

$$Dissimilarity(P1, P2) = 1 - Similarity(P1, P2) \tag{3.17}$$

Between any two consecutive frames, dissimilarity matrix of objects is made by the mentioned procedure and is given to the Hungarian assignment algorithm [26]. The output of this method is a list of objects that are connected to each other such that they result in the least total connections dissimilarity. By repeating the same procedure for all pairs of consecutive frames of the whole movie, we obtained all the tracks as sequences of connections.

3.3.2 A Multi-Frame Tracking Algorithm

For a Multi-Frame tracking algorithm, we used the method developed in [30] to find the K-Shortest-Paths (KSP) in a Flow Network with a greedy approach. This method is a state of the art multi-frame tracking approach. The goal is to find KSP from birth node to sink node. In this method, each detected object in any frame is considered as a node in the network. Only nodes of consecutive frames are connected to each other. We set the weight of each edge as the dissimilarity between the two nodes of the edge. Since each node in the network is an object, we used the same approach as the one we explained in Section 3.3.1 to measure the dissimilarity between the nodes. All the connections have a value between zero and one.

The other important factor in working with the Flow Network is the connections of the nodes to the birth and sink nodes. In slurry images, the objects usually sink. In other words, generally they appear at the top of the images and after moving downwards in a sequence of frames, they disappear at the bottom of the image. Therefore, connections to the birth and sink node in our network are related to their locations in the image, not their frame number. In each image, we connect the objects that are at the top part of the image to the birth node, as they are good candidates to start a track. Similarly, we connect the objects that are at the bottom of the images to the sink node as there is a good chance that those objects are observed for the last time in a sequence of images. To be more specific, the height of our images are 1500 pixels and the objects with a center's height less than 500 pixels (from the image top) are connected to the birth node and objects with center's height of more than 1000 pixels (from the image top) are connected to the sink node. The weight of each connection to the birth and sink node depends on the location of the object in the image. The higher (lower) the object is in the image, the higher the weight of connection to the birth (sink) node is. Weights are computed as follows:

$$Birth\_Weight(P) = 1 - \frac{Y(P)}{Birth\_Row} \tag{3.18}$$

$$Sink\_Weight(P) = 1 - \frac{Y(P) - Sink\_Row}{Image\_Height - Sink\_Row} \tag{3.19}$$

where $Birth\_Row = 500$, $Sink\_Row = 1000$, $Image\_Height = 1500$. $Birth\_Weight(P)$ and $Sink\_Weight(P)$ are the edge weights for the connections between object P and the birth node and the sink node, respectively.

We set up the network and assigned all the weight of edges as described before. Then we run the greedy algorithm described in [30] to find the K-Shortest-Paths (KSP) in the network. Since we were looking for all possible tracks, we set K to a large number ($K = 100$), meaning that the algorithm looks for a maximum of 100 paths with the shortest cost from the birth node to the sink node. In all of the algorithms, the total number of paths that were found was less than K. This indicates that all possible tracks without any shared paths are detected.

The results of evaluating two implemented tracking methods with regard to the ground truth tracks are shown in Figure 3.2 and Table 3.1. Here the results of Between Frame Based (BFB) evaluation and the algorithm running time are shown. The reported running time includes the time required for the tracking part of the algorithm only, and the running time of the object

Table 3.1: Between frame based evaluation of two tracking methods. Hungarian Assigning is a frame-by-frame tracking algorithm while finding KSP in Flow Network is multi-frame one. Because of using a greedy-based approach, finding KSP in Flow Network is significantly faster.

Between Frames Based Evaluation	**Precision_BFB**	**Recall_BFB**	**F_Score_BFB**	**Run Time (s)**
Hungarian Assigning	0.37	0.75	0.49	8.11
KSP in Flow Network	0.34	0.81	0.48	4.05

detection is excluded. In other words, the algorithm running time starts from when the information of the detected objects is extracted and it ends when all the tracks are identified by the tracking algorithm. For the Hungarian assigning approach, the running time of computing the dissimilarity matrix and finding the best matches between any two frames (starting from the first frame) are included. The run time is for the entire sequence. For the greedy Flow Network based method, running time includes the time required to make the entire network and then to find KSP in the network. Figure 3.2 shows the track based (TB) evaluation, where the y-axis shows the F-measure, and the x-axis shows different accuracy acceptance rates. Considering the F-measures and also the running time, it is clear that Flow Network based approach has a better performance.

3.4 Using Tracklets as Nodes in Flow Network

Section 3.3 indicates that finding KSP using Flow Network outperforms matching objects of consecutive images using Hungarian assigning method.

In this section, we examine whether using tracklets (a small part of a track) instead of objects as the nodes of the Flow Network would improve the performance of Flow Network-based tracking methods. In order to make the tracklets, there should exist an initial track to start with, and cut it into the tracklets. We generated the initial tracks by using Hungarian assigning method (See Section 3.3.1 for more detail). We then implemented and evaluated the performance of two existing track cutting methods (See Sections 3.4.1 and 3.4.2). In addition, we developed two novel cutting methods as explained in Sections 3.4.3 and 3.4.4 in more detail. Each of these methods takes a dif-

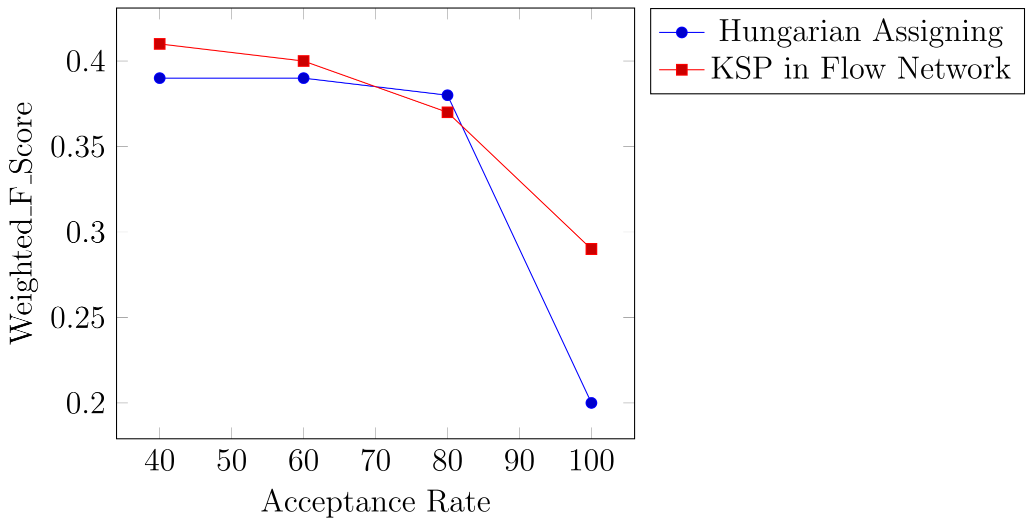


Figure 3.2: Track based evaluation of two tracking methods. Hungarian Assigning is a frame-by-frame tracking algorithm while finding KSP in Flow Network is multi-frame one. Acceptance rate is the a threshold and Weighted_F_Score is the *weighted_F_Score_TB* both mentioned in Section 3.2.2.

ferent approach to cut the tracks into smaller parts. In the following sections, we describe each of the tracklets generating algorithms (or track cutting algorithms) separately. Also, we report the performance of each method in terms of running time and F-Score. Please note that the reported running time of each method includes the time required for running the Hungarian algorithm, in addition to the time required to cut the tracks into the tracklets.

3.4.1 Singh2008 Cutting Method

Considering a specific track that is generated by the Hungarian assigning approach, the Singh2008 Cutting Method [41] traces the track starting from its first object, and computes the similarity between pairs of the objects in consecutive frames as it moves along the track. Whenever the similarity between the two objects is less than a predefined threshold S, it cuts the track to make the first tracklet. It then continues tracing the rest of the track by starting from the next object and repeating the same procedure to make a new tracklet.

3.4.2 Shitrit2014 Cutting Method

Considering a specific track that is generated by the Hungarian assigning approach, the Shitrit2014 Cutting Method [39] traces the track starting from its first object, and as it moves along the track, it computes the distance between the object and the closest object from another track at the same frame. Whenever the distance between the two objects get smaller than a predefined threshold, the track is cut and a tracklet is formed. The same procedure repeats for the rest of the track until the track comes to its end.

The rationale behind this method is that in most of the cases, the misassignment of the objects happens when the two tracks get too close. In this situation, there are more candidate objects to be assigned to a track when the Hungarian assigning approach is running, and therefore, the chance of a wrong assignment is higher. By cutting a track into the tracklet at these points, the error rate reduces.

3.4.3 Proposed Hard Cutting Method

In addition to the State-of-the-Art cutting algorithms (i.e. Singh2008 and Shitrit2014), we proposed two novel cutting methods to cut a track into several tracklets. The first one cuts a track every K steps (i.e. frames). This algorithm results in $\lceil \frac{L}{K} \rceil$ tracklets, where L is the length of the track. Here all the tracklets have a length of K except the last one, the length of which is equal to the remainder of dividing L by K. Our proposed cutting approach, which we call it the Hard Cut method, does not use any a prior information about the tracks and the objects.

The rationale behind using the Hard Cut method is that by using a small K (in the order of three to five frames), a large number of tracklets with a small size are generated. This would ensure that the possibility of encountering wrong assignments of newly detected objects to the existing tracks by using Hungarian assigning method remains negligible. Moreover, if a wrong assigning happens in the process, its effect size is small, as it remains in effect only by the end of the corresponding short tracklet. Given that our goal is to use the tracklets as the Flow Network objects, this method of cutting the tracks is a good candidate for our study, as it generates short reliable tracklets.

3.4.4 Proposed Kalman Filter Cutting Method

As our second proposed cutting method, we used Kalman filter to determine whether a big unexpected jump in the object position occurs on the track. In this method, a track is cut into smaller parts (i.e. tracklets) whenever the observed position of the current object is significantly far from its expected position defined by considering the object previous positions and speed. Different objects have different speeds. Moreover, the speed of each object varies as they move. As a result, setting a constant speed threshold (maximum number of pixels that an object moves between the two consecutive frames) is not appropriate for detecting unexpected big jumps in the location of the object.

As an alternative solution, we used the previous positions of the object to predict its expected position in the current frame. If the distance between the observed position and the predicted position is more than M times higher than the average distances (between the observed and predicted positions of the object) in the previous frames, then it is considered as a big jump, and the track is cut at that point. Using this approach, the definition of a big jump is specific to each track, and it is not dependent on a constant threshold.

More formally, we defined the distance between the predicted position of the object in track T in the ith frame and its observed location as:

$$D(T, i) = d(P(T, i), O(T, i)) \tag{3.20}$$

where $O(T, i)$ and $P(T, i)$ are the observed position and the predicted position (estimated by the Kalman Filter method) of the object in the ith frame of the track T, respectively. Here $d(P(T, i), O(T, i))$ is the Euclidean distance between these two positions. A big jump happens when the distance between the predicted position of the object and its actual observed position in the current frame is more than their average distance in the previous frames by a threshold fold of M:

$$D(T,i) > M * \frac{\sum\limits_{l=1}^{i-1} D(T,l)}{i-1} \tag{3.21}$$

If this condition is met, the algorithm cuts the track T at the ith frame, and continues to apply the same cutting process to the rest of the track. In this algorithm, the minimum length of a track is set to three frames, because the Kalman filter requires having some prior knowledge about the position of the object. Kalman filter is used to predict the position of the object in the fourth frame and afterward. Any track or the remaining part of a track with a length of fewer than three frames is considered as a tracklet by itself, and the Kalman filter is not applied to it.

In the Kalman filter, predicted state vector x_k^- and predicted covariance of state vector estimate P_k^- are computed in every iteration k by the following equations [44]:

$$x_k^- = Ax_{k-1} + Bu \tag{3.22}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.23}$$

where A is the state transition matrix, x_{k-1} is the prior state vector computed in the previous iteration (i.e. the predicted object location in the previous frame), B is input matrix, u is input control vector, P_{k-1} is covariance of state vector estimate in the last iteration, and Q is process noise covariance. The next step in each iteration is to update the x_k and P_k according to the newly observed location of the object and its related optimal Kalman gain K:

$$K = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3.24}$$

$$x_k = x_k^- + K(z_k - Hx_k^-) \tag{3.25}$$

$$P_k = P_k^- - KHP_k^- \tag{3.26}$$

where H is the observation matrix, R is the measurement noise covariance and z_k is the observation vector in k_{th} iteration (i.e. the location of the object in the current frame). In each iteration, x_k is the estimated location of the object in the next frame (i.e. $P(T, i)$ in Equation 3.20). Details about initializing the Kalman filter vectors and matrices is mentioned in the next section.

3.4.5 Setting Parameters of the Track Cutting Methods

For all of the above cutting methods, there was a parameter that needed to be set properly. The parameter S in the Singh2008 is the threshold of dissimilarity between the two consecutive objects in a track. The parameter D in the Shitrit2014 is the minimum distance between an object of one track and the objects of another track. The parameter K in the Hard Cut is the length of the tracklets, and the parameter M in Kalman Filter method is the fold threshold of current distance between the predicted and the observed positions of an object in the current frame compared to the average distance in the previous frames. In order to obtain the optimum value of the parameter for each cutting algorithm, we tested different numbers and selected the number that resulted in the best performance measured by Weighted F-Score according to between frame based (BFB) evaluation. The best parameter values are: $S = 0.2$, $D = 200$, $K = 3$ and $M = 2.75$.

To work with Kalman filter, several parameters are needed to be initialized including: x, state vector estimate, R, measurement noise covariance, H, observation matrix, Q, process noise covariance, P, covariance of state vector estimate, A, state transition matrix, B, input matrix and u, input control vector. In the implemented Kalman filter, state vector x is based on the location of the object and contains estimated X_Coordinate and Y_Coordinate of object location. z is the observation vector and it contains the current location of the object in the image. Since first state estimate is not available, x is initialized to be the first observed location. Parameters R, H, Q, A and Bu can have different values in each iteration but we assumed they are constant [44]. Following parameters are set according to object movement features in slurry images:

$$R = \begin{bmatrix} 0.2845 & 0.0045 \\ 0.0045 & 0.0455 \end{bmatrix}, H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, Q = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$
$$P = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}, A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Bu = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 6 \end{bmatrix} \tag{3.27}$$

After generating the tracklets, we set them as the nodes of the Flow Network. The nodes (i.e. Tracklets) that end in a specific frame were connected only to the tracklets that start in the same frame. Tracklets with their first object at the top part of the images were connected to the birth node and tracklets with their last object in the bottom part of the image were connected to the sink node. The weight of each connection to the birth node is computed based on the location of the first object in the tracklets. Similarly, The weight of each connection to the sink node is computed based on the location of the last object in the tracklets.

Computing the similarity between the nodes of the Flow Network was the same as what was explained in Section 3.3.1. The only difference is that instead of the shape and color of an object, we used the average shape (average number of pixels) and average color (average intensity) of objects in each tracklet. In addition, for computing the similarity between the positions of any two tracklets (Equation 3.14), the position of the last object of the previous tracklet was compared against the position of the first object of the current tracklet.

We measured the running time of each method. The total running time includes the time required to perform the following steps: **(1)** Making the initial tracks by using the Hungarian frame-by-frame assigning method; **(2)** Cutting the tracks into small tracklets; **(3)** Computing the tracklets information including average number of pixels and average intensity; and **(4)** Building the Flow Network according to the new features and finding the KSP in the network.

Table 3.2: Between frame based evaluation of tracking methods that use Flow Network, with or without tracklets. Methods which use tracklets need to have some tracks at the beginning, in order to cut them into trakclets. As a result, they are slower than original Flow Network.

Between Frames Based Evaluation	**Precision_BFB**	**Recall_BFB**	**F_Score_BFB**	**Run Time (s)**
Flow_Network_All_Graph	0.34	0.81	0.48	4.05
Flow_Network_Singh2008_Cut	0.47	0.65	0.54	10.1
Flow_Network_Shitrit2014_Cut	0.34	0.77	0.48	10.99
Flow_Network_Every_3_Steps_Cut	0.40	0.83	0.54	15.36
Flow_Network_Kalman_Filter_Cut	0.38	0.75	0.51	10.05

We evaluated the performance of the cutting methods using between frame based (BFB) and the track based (TB) evaluations methods, the results of which are shown in the Table 3.2 and Figure 3.3, respectively.

My results show that using the tracklets (instead of the detected objects) as the nodes of the Flow Network improves the performance of the KSP method. To be more specific, the Singh2008 cutting method and our two proposed cutting methods (Hard Cut and Kalman Filter Cut) had an improved performance compared to the original full Flow Network method (without cutting). The only cutting method that did not improve the results was the Shitrit2014. Based on BFB evaluations, the Singh2008 and the Hard Cut mutually show the best performance among the four tested cutting methods, while based on the TB evaluation, the best performance belongs to Singh2008 approach. The running times of the tested cutting methods were similar except for the Hard Cut approach which had the highest running time among the four methods. The main reason is that this method makes noticeably more tracklets, and therefore, computing the features of each tracklet and setting up the Flow Network take more time in the Hard Cut algorithm.

As it is clear from the Table 3.2, BFB Recalls of three out of four cutting methods are higher than 0.75; while the BFB precisions are relatively low (0.34-0.47). The low precisions resulted in low BFB F-Scores (less than 0.54). A low precision corresponds to a high false discovery rate (FDR), meaning that there are cases that the automated tracking algorithm identifies a track; while the manually identified ground truth does not include those tracks (False Positives). The main reason for the existence of many false positives is that

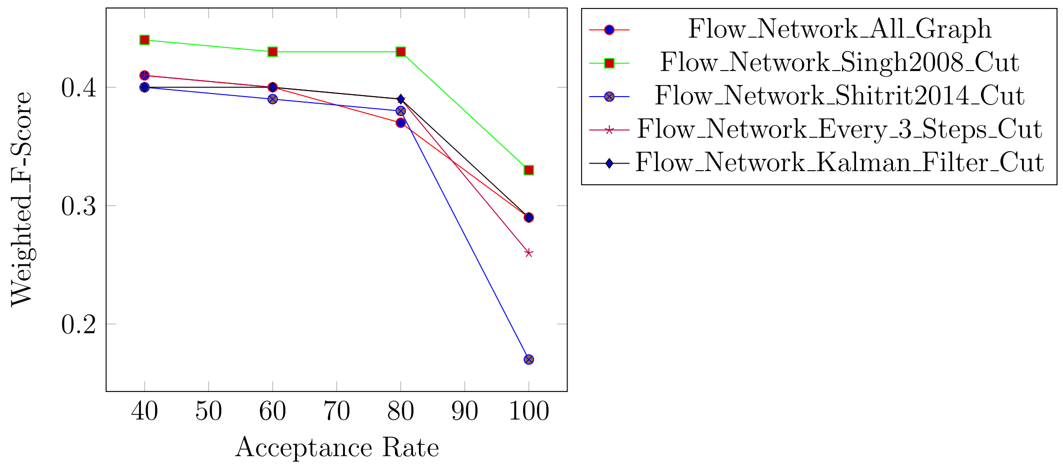


Figure 3.3: Track based evaluation of tracking methods that use Flow Network, with or without tracklets. All graph is the method in which every detected object is a node in the network, while tracklets are used instead in other four methods with different cutting algorithms. Acceptance rate is the a threshold and Weighted_F_Score is the $weighted\_F\_Score\_TB$ both mentioned in Section 3.2.2.

there are many similar and small objects in each image that make the task of generating the ground truth tracks hard for the user. In other words, the user was not able to consider some objects as part of a track by the bare eyes, while the fact that the tracking algorithms were able to identify those tracks may indicate that they were, in fact, the true tracks. Therefore, by missing to mark some of the true tracks as the ground truth tracks, the precision of the tracking algorithms were affected. This, in turn, resulted in relatively low F-Scores, although the BFB Recalls were relatively high. To get an idea of how a low precision may affect the F-Score, consider a case where the best possible correct answer (given that the user lost tracking of several tracks) results in a BFB Precision and Recall of 0.5 and 1, respectively. In this case, the best possible BFB F-Score would be:

$$F\_Score\_BFB = 2 * \frac{0.5 * 1}{0.5 + 1} = 0.66 \tag{3.28}$$

User limitation in selecting all the correct tracks affects the track-based (TB) evaluations too (See Figure 3.3). In TB evaluation, only tracks with an accuracy of $A(i) > t$ are considered for the evaluations. Here the accuracy of a track is equal to the number of frames that the track matches to any ground truth track divided by the length of that track. By missing some true tracks by the user, the accuracy of several tracks detected by an automated tracking algorithm falls below the threshold t. Therefore, according to the Equations 3.8 and 3.9, both precision, and recall measurements are effected and TB F-Score would be even less than BFB F-Score.

3.5 Computing Velocity of Bitumen Droplets

By detecting objects in each image and tracking the dark objects in sequences of images now we have the required information in order to compute the velocity of the bitumen droplets. At the end of tracking process, each tracking algorithm would give us several tracks. For each track, the velocity is computed as the average distance (number of pixels) traversed between consecutive frames. In general velocity is a 2D vector in our images, but since we know that objects do not move at the same pace in X and Y direction, therefore we focused on two 1D velocity vectors; **(1)** Vertical velocity which is the average number of pixels an object could move up and down between two frames and **(2)** Horizontal velocity which is the average number of pixels an object could move left and right between consecutive frames.

By having the velocity of objects (that are mostly bitumen droplets) we can also compute the relation between size or intensity of objects and their velocities. The scatter plots of the relation between size or intensity of the bitumen droplets and their horizontal and vertical velocities are shown in Figure 3.4 and Figure 3.5. Each node in these plots represents a track. The horizontal axis in Figure 3.4 is the average size of the object during the movement on the track and in Figure 3.5 is the average grayscale intensity of the object. In both figures, the vertical axis is for the average velocity of the object between consecutive frames. In these plots, results of two different tracking algorithms are shown. Black dots are the tracks computed by finding KSP in Flow Network with help of Singh2008 cutting algorithm and red dots represent the tracks of finding KSP in Flow Network when every detected object is a separate node in the network. Moreover, the relations between horizontal and vertical velocity and also between size of the objects and their intensity are shown in Figure 3.6.

According to the plots, most of the bitumen droplets have a size between 200 to 600 pixels and bigger objects move slower. On the other hand, there is a direct linear relation between the intensity of bitumen droplets and their velocity. Darker objects move very slowly and brighter objects move faster. We can also conclude that faster objects move fast in both directions and bigger objects are darker ones. In this project, our focus was on tracking darker detected objects (i.e. bitumen droplets) and therefore we do not have enough information about the relation between size and brightness of sand particles and their velocity.

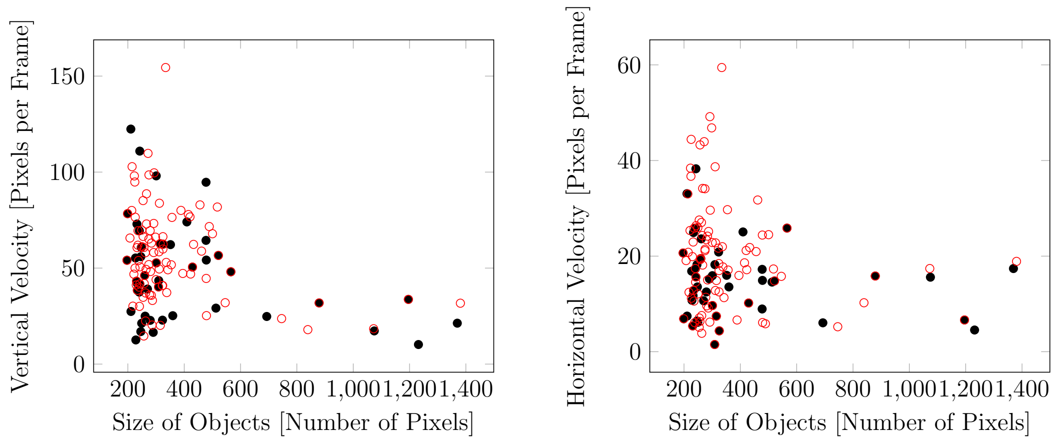


Figure 3.4: The relation between size of the objects and their velocity. Black dots represent 48 tracks computed by finding the KSP in Flow Network with help of the cutting algorithm used in Singh2008 for creating tracklets. Red circles are for 94 tracks computed by finding the KSP in Flow Network when every detected object is a node in the Flow Network.

3.6 Conclusion

In this chapter, we studied the process of tracking the detected objects in a sequence of slurry images. Due to the fact that there are a huge number of objects in each slurry image, the task of tracking the objects is very challenging. Since we are especially interested in computing the speed of bitumen droplets, the main target objects of our tracking method were the bitumen droplets,

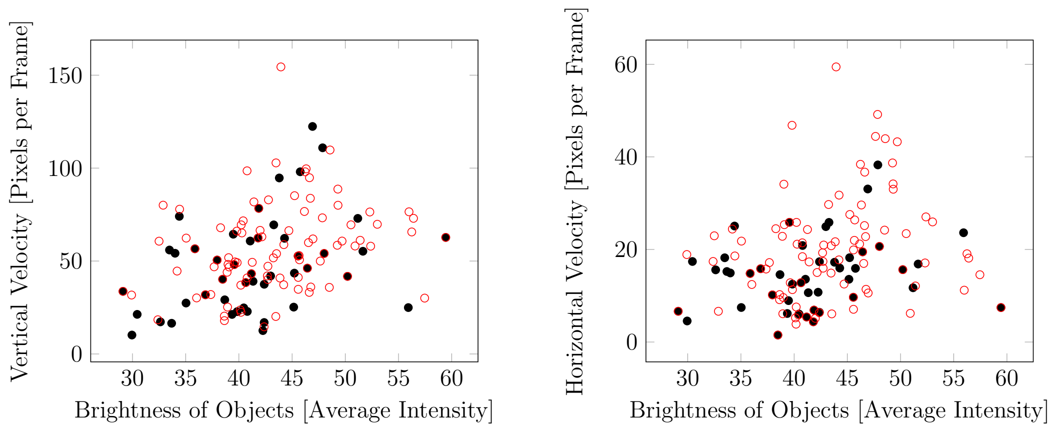
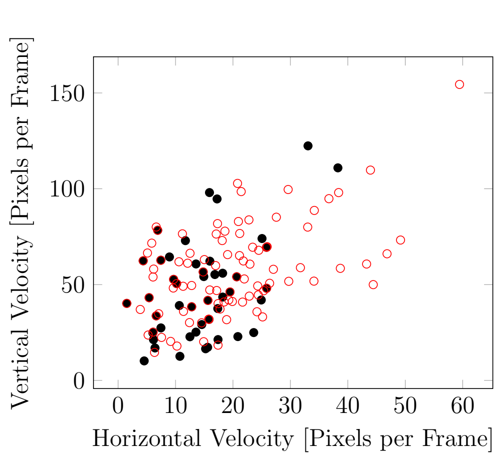


Figure 3.5: The relation between brightness of the objects and their velocity. Black dots represent 48 tracks computed by finding the KSP in Flow Network with help of the cutting algorithm used in Singh2008 for creating tracklets. Red circles are for 94 tracks computed by finding the KSP in Flow Network when every detected object is a node in the Flow Network.

which were the dark objects in the slurry images.

We first implemented and evaluated two tracking methods: (1) frame-by-frame based tracking; and (2) muli-frame based tracking. For the frame-by-frame based tracking algorithm, we used the Hungarian assigning algorithm to connect the corresponding objects in consecutive frames. At the end, the connected objects in the sequence of frames formed the tracks. For the muli-frame tracking method, we used a greedy-based algorithm to find the KSP in a Flow Network. Here every detected object was a node in the network, and the nodes of consecutive frames were connected to each other to form the network. The weights of the edges were computed based on the similarity between the related objects. In order to find the K best tracks in sequences of images, K-shortest Paths (KSP) should be found in this network. Our experiment results indicated that multi-frame-based approach using the Flow Network has better performance than the frame-by-frame method, and its running time is noticeably shorter.

In order to evaluate the performance of the tracking methods, we needed to have ground truth tracks. Therefore, we implemented a ground truth generator GUI. This GUI helps a user select a certain detected object in a sequence of images and make the ground truth tracks fast and accurately. Using the ground



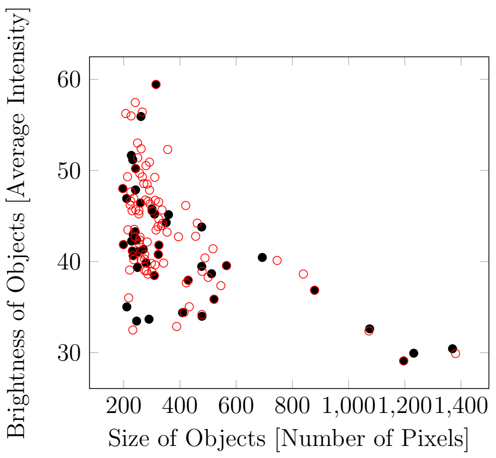


Figure 3.6: Horizontal velocity vs Vertical velocity and Size vs Brightness. The relation between horizontal velocity and vertical velocity is shown in the left plot, and the relation between size of the objects and their brightness is shown in the right plot. Black dots represent 48 tracks computed by finding the KSP in Flow Network with help of the cutting algorithm used in Singh2008 for creating tracklets. Red circles are for 94 tracks computed by finding the KSP in Flow Network when every detected object is a node in the Flow Network.

truths dataset, we measured evaluation metrics such as precision, recall, and F-score. Two different ways of evaluations are described in Section 3.2: between frames based evaluation (BFB) and track based evaluation (TB). In all the evaluations, tracks should start at the top of the image and finish at the bottom of the image. Situations such as merging or splitting of the objects are not in the scope of our study and are left for the future work.

Realizing that finding KSP in Flow Network leads to a better performance compared to the frame-by-frame approaches, we took the next step and asked whether we can improve the results even further. According to the literature, an appropriate way to have more reliable features for the Multiple Target Tracking (MTT) methods is to use the tracklets instead of the original objects in the Flow Network. Tracklets are small sub-tracks with short lengths. Using tracklets helps to extract more reliable information for each object such as its average size and color during its path in the tracklet. Here our main aim was to use the tracklets instead of each object separately as the nodes of the Flow Network and to make the connections between timely matched tracklets (the last object of previous tracklet and the first object of current tracklet should be in consecutive frames). The dissimilarity between the Flow Network nodes was measured based on the dissimilarity between the tracklets.

We implemented two existing methods and developed two novel methods to generate the tracklets, and to evaluate the effects of using tracklets on the performance of the Flow Networks in tracking the objects. Our first proposed method (Hard Cut) divides the tracks into smaller tracklets every K frame without any prior information about tracks; while our second proposed method (Kalman Filter), cuts a track wherever a big unexpected jump is detected on the track by a Kalman Filter based approach. Results of evaluations indicate that using the tracklets improves the performance when these tracklets were generated by any of the Singh2008, Hard Cut, or Kalman Filter methods. The only cutting method that did not improve the results was Shitrit2014. Based on BFB evaluations, the Singh2008 and the Hard Cut mutually show the best performance among the four tested cutting methods, while based on the TB evaluation, the best results were obtained by Singh2008 approach. Overall Kalman Filter approach is faster than Hard Cut while both of them have similar performances. By having the track results, we computed the relation between size and brightness of objects and their velocity. According to the results, brighter and smaller bitumen droplets are faster.

Chapter 4

Summary, Conclusion and Future Works

Tracking several objects in an environment is a well-known image processing and vision problem that has applications in several areas including blood cell tracking, pedestrian tracking and analyzing players of a game. In this study, we have addressed this problem for a specific application: Computing speed of bitumen droplets in a sequence of slurry images in the real-time. To achieve this, we developed algorithms performing the following two steps: (1) detecting as many objects (i.e. bitumen droplets and sand particles) as possible in each image in the shortest running time; and (2) tracking the detected objects in the video. In our study, the first priority was given to tracking bitumen droplets. Our study improved the performance of both detection and tracking algorithms for the current application.

4.1 Detecting Objects in Each Image

Chapter 2 of this thesis is dedicated to applying different segmentation and thresholding algorithms to slurry images with the aim of detecting objects including bitumen droplets and the sand particles in these images.

Here the goal was to develop a method to detect as many objects as possible with low false detection rate. The desired method should process the images in the real-time. In order to identify the best detection method, we built a ground truth data through manually labeling ten slurry images from different moments of floating slurry in the flow tube. We then evaluated the performance of four global thresholding, seven local thresholding and five segmentation algorithms using this ground truth data. Our results indicated that MAX-ENT and NICK outperform the other algorithms in both sparse and dense images. We verified the effect of tiling and downsampling of images when combined with all the mentioned algorithms. Our results showed that the tiling and downsampling of images did not improve the quality of object detection. Although tiling and downsampling could decrease the running time of segmentation algorithms but they are still not as fast as thresholding methods.

My other contribution to this study was developing a recursive iterative thresholding framework that is applicable to both global and local thresholding algorithms and results in detection of a larger number of small and bright objects in each image. Our recursive iterative thresholding framework uses a thresholding algorithm and iteratively applies the following steps for each image until the termination condition is met: (1) detect objects using a thresholding algorithm; (2) select dark objects and add them to the output image; (3) remove the selected objects from the original image; (4) repeat steps 1-3 if the termination condition is not met.

In order to evaluate the performance of our recursive iterative thresholding framework, we applied it in combination with the best local and global thresholding algorithms (MAX-ENT and NICK) to our slurry images. Our results indicated that this framework is capable of detecting more objects in each image including small and bright objects. Our recursive iterative framework improved the performance of both tested thresholding algorithms with the cost of longer running times.

4.2 Tracking Objects in a Sequence of Images

After detecting bitumen droplets and sand particles properly, the next step is to track the detected objects in a sequence of images. Chapter 3 of this thesis is dedicated to explaining the computational methods we developed to track bitumen droplets in a sequence of slurry images. Here the goal was to develop a method to track bitumen droplets fast and accurately.

In our study, we considered two approaches for tracking the objects: (1) Hungarian-based frame-by-frame tracking approach; and (2) Flow Network-based tracking approach. In order to evaluate each method, we built the ground-truth tracks through developing a GUI that helps a user to select the tracks fast and easy. We then used the ground-truth tracks to evaluate the performance of the Hungarian-based and the Flow Network-based tracking approaches. Our results indicated that the greedy based implementation of finding K-shortest path (KSP) in a Flow Network outperforms the Hungarian-based approach in terms of both performance and the running time.

In our initial Flow Network-based experiments, we used each detected object as a node of the network. We then asked if using tracklets (i.e. the short tracks, most of the time obtained by cutting a long track) instead of objects as the nodes of the network would improve the performance of Flow Network-based method.

In order to make tracklets, we used two existing cutting algorithms (Shitrit2014 and Singh2008) [39, 41]. In addition, we proposed two cutting methods and called them Hard Cut and Kalman Filter Cut. The Hard Cut algorithm divides a track into smaller tracklets with the fixed length, while the Kalman Filter Cut algorithm cuts a track whenever there is a big jump in the location of the object in a track as the object moves in a sequence of images. Our results indicated that using tracklets improves the performance of finding KSP in the Flow Network. In addition, we showed that although Singh2008 [41] outperforms all other approaches, but our proposed methods in generating tracklets have better performances than the other State-of-the-Art of cutting algorithm Shitrit2014 [39].

As the final experiment, we were interested in knowing the relation between the size and the brightness of bitumen droplets and their velocity. The result indicated that larger bitumen droplets move slower than the smaller ones and moreover, brighter objects move faster than darker objects.

4.3 Future Works

We would suggest the following directions for the future work in order to continue this research study and to improve the results:

- **Using cameras with higher frame rates for taking videos of the sinking process.** In the current videos, positions of the objects change substantially in the consecutive frames, making it very challenging to track them. Using higher frame rates cameras would result in more accurate tracking of the objects, which in turn results in more accurate computing of the velocity of the objects.

- **Applying more tracking algorithms on slurry images, and evaluating their results.** In this study, we used two tracking algorithms. As a future direction, we would suggest using other approaches such as Gmcp-tracker [48] and multiple hypothesis tracking (MHT) [5] as they both showed good performance in tracking multiple targets in other MTT scenarios such as pedestrian tracking.

- **Improving quality and quantity of the ground truth datasets.** Generating higher number of the labeled ground truth images for the object detection stage of the framework, as well as, generating more ground truth tracks using our GUI in the other similar videos would increase the accuracy of our evaluations of the object detection and tracking methods.

- **Applying object detection and tracking algorithms to other datasets.** Although we applied our methods to this specific application (i.e. identifying and tracking bitumen droplets in slurry images), our framework is general and can be used in other applications that include images with similar characteristics. An example of these applications is tracking the blood cells in a sequence of images. We am also keen to see how our proposed cutting approaches and recursive iterative thresholding method perform in other applications.

- **Extending the method to detect and track more diverse types of objects.** In slurry images, some objects may merge together or split from each other as they move. The merging and splitting effects induce more challenges in tracking the objects, as they may change the features of the objects such as their sizes and shapes. Our current method does not address these cases. However, this is a very interesting direction for the future work, as it improves the quality of object tracking in this specific application, and we predict it will find many applications in the other similar datasets. Moreover, in order to extract more information about floating objects and the slurry process, there is a need to track a larger number of objects including sand particles. In this project, our focus and priority were on tracking bitumen droplets but for further studies, tracking sand particles is recommended.

Bibliography

[1] Ahmed S Abutaleb. Automatic thresholding of gray-level pictures using two-dimensional entropy. *Computer vision, graphics, and image processing*, 47(1):22–32, 1989.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[3] Rolf Adams and Leanne Bischof. Seeded region growing. *IEEE Transactions on pattern analysis and machine intelligence*, 16(6):641–647, 1994.

[4] John Bernsen. Dynamic thresholding of grey-level images. In *International conference on pattern recognition*, volume 2, pages 1251–1255, 1986.

[5] Samuel S Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, 2004.

[6] Gregory Castñnón and Lucas Finn. Multi-target tracklet stitching through network flows. In *Aerospace Conference, 2011 IEEE*, pages 1–7. IEEE, 2011.

[7] Nicolas Chenouard, Ihor Smal, Fabrice De Chaumont, Martin Maška, Ivo F Sbalzarini, Yuanhao Gong, Janick Cardinale, Craig Carthel, Stefano Coraluppi, Mark Winter, et al. Objective comparison of particle tracking methods. *Nature methods*, 11(3):281, 2014.

[8] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[9] John C Crocker and David G Grier. Methods of digital video microscopy for colloidal studies. *Journal of colloid and interface science*, 179(1):298–310, 1996.

[10] Darcy Daugela, Barry Bara, Robert Skwarok, Rodney Ridley, Pat Dougan, and Mark Polak. System and method for image-based analysis of a slurry and control of a slurry process, March 24 2016. US Patent 20,160,086,321.

[11] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[12] Basilios Gatos, Ioannis Pratikakis, and Stavros J Perantonis. Adaptive degraded document image binarization. *Pattern recognition*, 39(3):317–327, 2006.

[13] Khuloud Jaqaman and Gaudenz Danuser. Computational image analysis of cellular dynamics: a case study based on particle tracking. *Cold Spring Harbor Protocols*, 2009(12):pdb–top65, 2009.

[14] Khuloud Jaqaman, Dinah Loerke, Marcel Mettlen, Hirotaka Kuwata, Sergio Grinstein, Sandra L Schmid, and Gaudenz Danuser. Robust single-particle tracking in live-cell time-lapse sequences. *Nature methods*, 5(8):695–702, 2008.

[15] Jagat Narain Kapur, Prasanna K Sahoo, and Andrew KC Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer vision, graphics, and image processing*, 29(3):273–285, 1985.

[16] Muhammad Burhan Khan, Humaira Nisar, Choon Aun Ng, Po Kim Lo, and Vooi Voon Yap. Local adaptive approach toward segmentation of microscopic images of activated sludge flocs. *Journal of Electronic Imaging*, 24(6):061102–061102, 2015.

[17] Rohollah Mazrae Khoshki and Subramaniam Ganesan. Multi-scale adaptive nick thresholding method for alpr system. *Entropy*, 4(10), 2015.

[18] Khurram Khurshid, Imran Siddiqi, Claudie Faure, and Nicole Vincent. Comparison of niblack inspired binarization methods for ancient documents. In *IS&T/SPIE Electronic Imaging*, pages 72470U–72470U. International Society for Optics and Photonics, 2009.

[19] Josef Kittler and John Illingworth. Minimum error thresholding. *Pattern recognition*, 19(1):41–47, 1986.

[20] Klas EG Magnusson, Joakim Jaldén, Penney M Gilbert, and Helen M Blau. Global linking of cell tracks using the viterbi algorithm. *IEEE transactions on medical imaging*, 34(4):911–929, 2015.

[21] GI Mashanov and JE Molloy. Automatic detection of single fluorophores in live cells. *Biophysical journal*, 92(6):2199–2211, 2007.

[22] Martin Maška, Vladimír Ulman, David Svoboda, Pavel Matula, Petr Matula, Cristina Ederra, Ainhoa Urbiola, Tomás España, Subramanian Venkatesan, Deepak MW Balak, et al. A benchmark for comparison of cell tracking algorithms. *Bioinformatics*, 30(11):1609–1617, 2014.

[23] Jacob Masliyah, Zhiang Joe Zhou, Zhenghe Xu, Jan Czarnecki, and Hassan Hamza. Understanding water-based bitumen extraction from athabasca oil sands. *The Canadian Journal of Chemical Engineering*, 82(4):628–654, 2004.

[24] Erik Meijering, Oleh Dzyubachyk, Ihor Smal, et al. Methods for cell and particle tracking. *Methods Enzymol*, 504(9):183–200, 2012.

[25] Reza Farrahi Moghaddam and Mohamed Cheriet. Adotsu: An adaptive and parameterless generalization of otsu's method for document image binarization. *Pattern Recognition*, 45(6):2419–2431, 2012.

[26] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[27] Wayne Niblack. *An introduction to digital image processing*. Strandberg Publishing Company, 1985.

[28] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1452–1458, 2004.

[29] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[30] Hamed Pirsiavash, Deva Ramanan, and Charless C Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1201–1208. IEEE, 2011.

[31] Ioannis Pratikakis, Basilis Gatos, and Konstantinos Ntirogiannis. Icdar 2013 document image binarization contest (dibco 2013). In *2013 12th International Conference on Document Analysis and Recognition*, pages 1471–1476. IEEE, 2013.

[32] TW Ridler and S Calvard. Picture thresholding using an iterative selection method. *IEEE trans syst Man Cybern*, 8(8):630–632, 1978.

[33] Salman S Rogers, Thomas A Waigh, Xiubo Zhao, and Jian R Lu. Precise particle tracking against a complicated background: polynomial fitting with gaussian weight. *Physical Biology*, 4(3):220, 2007.

[34] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.

[35] Ivo F Sbalzarini and Petros Koumoutsakos. Feature point tracking and trajectory analysis for video imaging in cell biology. *Journal of structural biology*, 151(2):182–195, 2005.

[36] Arnauld Sergé, Nicolas Bertaux, Hervé Rigneault, and Didier Marguet. Dynamic multiple-target tracing to probe spatiotemporal cartography of cell membranes. *Nature methods*, 5(8):687–694, 2008.

[37] Mehmet Sezgin et al. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–168, 2004.

[38] Faisal Shafait, Daniel Keysers, and Thomas M Breuel. Efficient implementation of local adaptive thresholding techniques using integral images. In *Electronic Imaging 2008*, pages 681510–681510. International Society for Optics and Photonics, 2008.

[39] Horesh Ben Shitrit, Jérôme Berclaz, François Fleuret, and Pascal Fua. Multi-commodity network flow for tracking multiple people. *IEEE transactions on pattern analysis and machine intelligence*, 36(8):1614–1627, 2014.

[40] T Romen Singh, Sudipta Roy, O Imocha Singh, Tejmani Sinam, Kh Singh, et al. A new local adaptive thresholding technique in binarization. *arXiv preprint arXiv:1201.5227*, 2012.

[41] Vivek Kumar Singh, Bo Wu, and Ramakant Nevatia. Pedestrian tracking by associating tracklets using detection residuals. In *Motion and video Computing, 2008. WMVC 2008. IEEE Workshop on*, pages 1–8. IEEE, 2008.

[42] Matthew B Smith, Erdem Karatekin, Andrea Gohlke, Hiroaki Mizuno, Naoki Watanabe, and Dimitrios Vavylonis. Interactive, computer-assisted tracking of speckle trajectories in fluorescence microscopy: application to actin polymerization and membrane fusion. *Biophysical journal*, 101(7):1794–1804, 2011.

[43] Engin Türetken, Xinchao Wang, Carlos Joaquin Becker, Carsten Haubold, and Pascal Fua. Globally optimal cell tracking using integer programming. Technical report, 2016.

[44] Greg Welch and Gary Bishop. An introduction to the kalman filter. department of computer science, university of north carolina, 2006.

[45] Pierre D Wellner. Adaptive thresholding for the digitaldesk. *Xerox, EPC1993-110*, 1993.

[46] Junliang Xing, Haizhou Ai, and Shihong Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1200–1207. IEEE, 2009.

[47] Feixiang Yan, Hong Zhang, and C Ronald Kube. A multistage adaptive thresholding method. *Pattern recognition letters*, 26(8):1183–1191, 2005.

[48] Amir Roshan Zamir, Afshin Dehghan, and Mubarak Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. In *Computer Vision–ECCV 2012*, pages 343–356. Springer, 2012.

[49] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[50] Zhou Zhiwei, Li Linlin, and Tan Chew Lim. Edge based binarization for video text images. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 133–136. IEEE, 2010.