DEVELOPMENT OF A HEART MOTION TRACKING SYSTEM USING NON-INVASIVE IMAGING DATA

by

Bryent Tucker

July, 2017

Director of Thesis: Dr. Zhen Zhu

Major Department: Engineering

Cardiac motion can be monitored non-invasively for the assessment of cardiovascular function by using medical imaging systems and motion tracking algorithms. Existing tracking approaches require *a priori* understanding of the non-rigid motion of the target system, which could change over multiple cardiac cycles and lead to tracking failures. The purpose of this research is to develop the algorithm and software, with computer vision techniques, to continuously track the motion of a user-defined region of the heart images. The proposed algorithm improves upon existing techniques because it does not require an underlying motion model, it quantifies the quality of tracking, and it can recover from a failed tracking estimate. The motion estimation of a non-rigid system will be done by a piecewise tracking approach that breaks up the region of interest into several small segments (patches), which can be approximated with interconnected pseudo-rigid segments. These segments will be initialized based on two criteria: 1) motion within a segment must follow the pseudo-rigid body model; and 2) motion in neighboring segments must be similar to each other. Segments are subsequently tracked as pseudo-rigid bodies, and the criteria described above are also used to detect failures in tracking. If a failure were to occur, the tracking algorithm will be reinitialized automatically. This algorithm was shown to be accurate and efficient, and has been tested on several heart motion data sets.

Development of a Heart Motion Tracking System using Non-invasive Imaging Data

A Thesis

Presented to the Faculty of the Department of Engineering

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science Biomedical Engineering

by

Bryent Tucker

July, 2017

Development of a Heart Motion Tracking System

using Non-invasive Imaging Data

by

Bryent Tucker

APPROVED BY:

DIRECTOR OF

THESIS: _____

(Zhen Zhu, PhD)

COMMITTEE MEMBER: _____

(Sunghan Kim, PhD)

COMMITTEE MEMBER: _____

(Jun Qing Lu, PhD)

CHAIR OF THE DEPARTMENT

OF (Engineering): _____

(Hayden Griffin, PhD)

DEAN OF THE

GRADUATE SCHOOL: _____

Paul J. Gemperline, PhD

ACKNOWLEDGEMENTS

I would first like to thank Dr. Ferguson and Dr. Chen for providing all five heart motion data sets. This data provided the support in the development and testing of the heart motion tracking system. I would also like to thank my thesis advisor, Dr. Zhu, along with my committee members, Dr. Lu and Dr. Kim, for their tremendous help in guiding me throughout this research process. I would lastly like to thank my fiancée, Amanda, and family for their consistent support as I completed this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CHAPTER 1: INTRODUCTION

Cardiovascular diseases are defined by the World Health Organization (WHO) as a group of disorders of the heart and blood vessels that restrict blood flow to a specific region. These diseases are also labeled as ischemic disorders that include coronary heart disease, peripheral arterial disease, and deep vein thrombosis. The disorders classified can lead to severe conditions that include myocardial infarctions, heart failure, strokes, and more. Cardiovascular diseases are the leading cause of death globally, and an estimated 17.5 million people died from these diseases in 2012 (WHO, 2016). Cardiac motion must be monitored non-invasively for the assessment of cardiovascular function using motion tracking systems and medical imaging data. The non-invasive tracking systems must be able to model and predict the movement of structures of the heart muscle, and analyze their movement overtime for applications that include diagnostic analysis.

Computer vision techniques have been widely used in the medical field to develop algorithms for these applications. For example, post-processing techniques have been used to improve the quality of images and to extract information. Some of these algorithms have been implemented in software to improve the overall image intensity, to analyze specific regions of interest, or to take measurements for accurate analysis and diagnosis (Najarian & Splinter, 2012). Some of the existing cardiac motion tracking techniques estimate the movement of structures of the heart muscle by using a specific model to predict this motion. Cardiac motion tracking by this approach requires *a priori* understanding of the motion. Since the motion of the heart could change over multiple heart cycles, the models developed could fail to account for this change, which makes it difficult to recover from tracking failures. The research presented in this work will aim at improving the heart motion tracking algorithms.

The purpose of this research is to develop the algorithm and software, by using computer vision techniques, to continuously track the motion of a biological heart from two-dimensional imaging data without using a motion model. The non-parametric motion tracking algorithm will be used to estimate the motion of non-rigid structures of the heart by tracking point features identified in a user-defined local region of interest (ROI), on or within the heart. The algorithm will be developed using multiple computer vision techniques, including, but not limited to, feature extraction, matching, motion estimation, and outlier detection techniques. It is fully

realized that point feature-based tracking methods could potentially be disrupted by large motions, noise, lost frames or occlusions. It is important for a tracking system to have the capability of detecting such conditions. In this method, the tracking quality of these point features will be quantified by evaluating the consistency in frame. If the tracking quality fails a set of criteria, the algorithm is able to flag the failure of tracking. Future, it will be able to realign with the motion and recover from the failure. Finally, it is purely based on two-dimensional imagery data, and is thus considered a non-invasive approach.

The developed algorithm from this thesis can be utilized in many other applications. For example, the movement of myocardial walls or blood vessels could be tracked continuously to assess function by analyzing the motion and strain fields of these structures. Another possible application for this algorithm could be long-term single-cell tracking, which could improve the analysis of the cellular dynamics of a specific cell line. Current research in this field looks to improve upon the lack of software techniques in single-cell tracking, and reduce the amount of data acquisition needed (Hilsenbeck et al., 2016). Since cells move as non-rigid bodies, the developed algorithm can be used in this application to highlight and continuously track a specified cell for analysis. Since most tissues and organs that make up the human body are non-rigid structures that need to be tracked continuously, an additional application for this algorithm could be in the diagnostic analysis of other structures using imaging modalities. If a measurement is to be acquired using a specified imaging technique, the algorithm will be able to track the non-rigid motion of a region to provide the supplemental geometry for continuous data acquisition.

The algorithm proposed in this thesis improves the existing techniques because 1) it does not require underlying motion model to be known, 2) it quantifies the current tracking estimate, and 3) it can recover from a failed tracking estimate. Chapter 2 will review the current motion tracking techniques that are applicable to biomedical systems and estimating cardiac motion. The underlying methods used for algorithm development and the experimental design will be presented in chapter 3. Lastly, the results of the tracking algorithm, discussion and a conclusion will be presented in chapters 4, 5, and 6 respectively.

CHAPTER 2: REVIEW OF THE LITERATURE

The literature review will focus on three areas. The first focus is on the research related to heart motion estimation and modeling using medical imaging data. The second part of this review is focused on motion tracking models developed for robotic-assisted surgery applications. Finally, the existing approaches that are related to the proposed algorithm of tracking non-rigid motion, such as piece-wise tracking, will be discussed.

2.1: Medical Image Tracking: MRI Tagging

Different imaging modalities, such as CT, MRI, and ultrasound, have been used as diagnostic tools to monitor patient cardiac function and analyze cardiac motion to improve diagnosis. MRI tagging has been a useful procedure to obtain detailed information to track the motion and strain fields of myocardium. MRI tagging was introduced in the late eighties, when Zerhouni et al. developed a technique for generating visible image markers to tack myocardial movement without the need for physically implanted markers (Zerhouni et al., 1988). Two of the most popular MRI tagging techniques used today are the spatial modulation of magnetization (Axel & Dougherty, 1989) and complementary spatial modulation of magnetization (Fischer et. al, 1993) labeled as SPAMM and CSPAMM respectively. Both techniques generate a grid pattern across the images for cardiac motion analysis. SPAMM generates this tag pattern through selective radio frequency (RF) pulses throughout the cardiac cycle, while CSPAMM improves upon this technique to reduce fading of the tag pattern and improve the signal-to-noise ratio (Ibrahim, 2011). Research has been completed on the development of automated tools and tracking models to assist clinicians with cardiac motion diagnosis, and three different examples of these tools will be discussed further.

In a study conducted by Chandrashekara et al. (2003), a statistical model of the myocardium motion field of several healthy volunteers' tagged MRI data was developed and tested. The objective of this research was to use prior information to develop a statistical model to track the motion of myocardium from patient data. To develop the statistical model, data from 17 volunteers, consisting of short-axis (SA) and long-axis (LA) slices, was acquired. Also, a cine breath-hold sequence of data was taken to develop a SPAMM tag pattern at the end of expiration for generating a grid pattern across the images to analyze motion (Chandrashekara et. al, 2003).

With the MRI tagged data collected, a statistical model was generated from all image frames between end-diastole and end-systole using all seventeen patients. Both the SA and LA data was used to derive a myocardial motion field for all individual subjects, and to account for the twisting, contraction, and shortening of the heart. The SA and LA images acquired were re-sampled by ordering images such that the first frame is aligned with end-diastole, or relaxation. With the images re-sampled to follow the cardiac cycle, a transformation function was derived to estimate the movement of any point in the myocardium at each frame. The actual motion fields for each image were then calculated by transforming multiple points within one image frame to estimate the position of those points in the next frame, and subtracting transformed position with the actual location in the current frame. The motion fields were computed to model the deformation of the heart within the image throughout time. Next, the motion fields of each data set were mapped temporally and spatially to a reference subject. This was done to develop a common coordinate system for points within each image and develop a common time scale for all image frames. With the common coordinate system developed, principal component analysis was conducted to build the statistical model for the cardiac motion. Two separate models were built and validated by tracking the motion of the heart in eight separate data sets for all time frames of the cardiac cycle. Both models differed in specific parameters, but the displacement of tag intersection points was compared to the displacement of these points measured by a human observer for each model. The root-mean-square tracking error was found to be below 2 millimeters for the cardiac cycle over all eight data sets, which is labeled as a good performance for motion tracking (Chandrashekara et. al, 2003).

The results of this study show that the cardiac motion can be modeled and applied to different data sets to track the motion of intersecting lines, or deformations, from tagged MRI data with a small displacement error. It further shows that the heart motion can be modeled and tracked overtime. The model was not optimal for real-time tracking, as it does not account for the variability of the heart rate over multiple contraction cycles. Improvements to the model could be made to account for the variation from changes in the motion pattern of different healthy and diseased patients. The method proposed in this study also requires the use of tagged MRI images, which increases data acquisition cost. The model was also not developed for applications that require tracking recovery from lost data or occlusions.

In a study conducted by Hassanein et al. (2014), a mathematical model was generated to test tracking methods of cardiac motion with synthetic images, to model left ventricular function of tagged MRI data. The synthetic data was modeled as a circular disc with an inner radius of 30 millimeters and outer radius of 40 millimeters to simulate the endocardium and epicardium. The displacement model varied over polar coordinates, where the radius decreased to a specific point and then increased to simulate the contraction and relaxation of the cardiac cycle. The polar coordinates were then transformed into a Cartesian system to develop a sequence of test images. Using a known model, the SPAMM and CSPAMM tag patterns were overlaid on the image to simulate actual tagged data. To complete the synthetic data, Gaussian white noise and exponential decay of the tag line amplitude was applied to simulate actual noise artifacts from real time imaging. The synthetic data set motion was then tracked overtime by comparing optical flow techniques with a commercially utilized HARP technique. The harmonic phase algorithm (HARP) is a commercially available MRI analysis technique to process the motion of tagged MRI data (Hassanein et. al, 2014).

Image tracking techniques based on optical flow were applied to the synthetic data set, along with HARP to estimate the motion of the circle over time. Optical flow is a computer vision technique to track the change in pixel motion overtime. Optical flow can be defined as the two-dimensional displacement or velocity estimation of pixel patches on an image plane. Three different optical flow techniques Lucas-Kanade (LK), Horn-Shuck (HS), and Anisotropic LK (LKD) were used in this study. The radial strain and estimation errors were calculated over each image frame for all four methods, and the radial strain was compared to the actual strain measured from the image sequences. The results show that the LK method produced the most accurate strain measurement at the epicardial or border of the circle for tag patterns, and produced the smallest tracking error overall for all tests. The commercially available method, HARP, was found to have good tracking accuracy in the endocardium or inner circle, but failed to track the border of the circle for both tag patterns. The HS results were found to have the greatest tracking error among optical flow techniques, but performed better than the HARP method at the border. The LKD tracking error was found to be similar to that of the LK method at the border, but was greater at the inner circle than the LK error (Hassanein et. al, 2014).

In a similar study, Arif et al. (2014) proposed a method for tracking specific structures in cardiac MRI images by propagating a segmentation from one frame to another across a sequence of images. This study improved upon the partial differential equation (PDE) solutions of optical flow techniques with consideration of the fluid motion present in the heart. The researchers proposed a new boundary condition to solve the PDE of the tracking methods by changing them from a zero velocity to a value linked to the inside of the segmentation that is not zero. The mathematical model developed was then discretized to generate an algorithm to be tested. The developed method was applied to publicly available data sets, the MICCAI Left and Right Ventricle sets (LV and RV), and compared to the commercially available Medivisio segmentation software. Both methods start with a user selected initial segmentation of each ventricle, and the segmentation is propagated separately to compare both techniques. Full heart segments were also selected to track the myocardium outer boundary, left ventricle, and right ventricle (Arif et. al, 2014).

The results show that the proposed technique from this study improves cardiac segmentation and propagation throughout the entire data set when compared to Medivisio segmentation, and requires no manual correction throughout the entire data set. For the LV, the proposed method included a more accurate boundary segmentation and motion compared to Medivisio. However, the main difference between the two can be seen in the RV data. The Medivisio segmentation would group noise artifacts and components outside the true boundary of the RV. This would lead to a correction by the user to delete extra unwanted pixels to continue tracking. The proposed method was not interrupted by these artifacts and kept an accurate boundary segmentation close to the actual RV wall. When these techniques were applied to segment out multiple structures, the Medivisio method failed and consistently could not correct itself to keep the segment boundary equal to the actual movement of the LV, RV, and myocardium wall. The proposed method accurately tracked the boundary of the segmented LV, RV, and myocardium wall throughout multiple cardiac cycles (Arif et. al, 2014).

By making improvements to the general optical flow techniques, the proposed method from this study demonstrated accurate segmentation and tracking results for LV, RV, and myocardial boundaries. The proposed technique was also developed to be computationally efficient because the changes made did not alter the computational time significantly, when

compared to traditional techniques (Arif et. al, 2014). The method did not even require specific MRI tagging, and was developed to be non-specific to any imaging modality. The method was not developed for real-time tracking as the experiments were conducted as post-processing techniques on MRI data that is publicly available and does not include interference or occlusions. The acquisition cost to use this technique for MRI data would lead to an expensive data acquisition for non-invasive diagnosis.

In summary, MRI is one of the medical imaging modalities used to track the motion of specific components of the heart, which requires preprocessed (tagged) data and can only track the heart within one region, such as the left ventricle. The high cost associated with MRI tagged data promotes the use of other imaging methods in the development of a tracking algorithm for diagnostic applications. The tracking algorithms presented support the use of optical flow techniques to estimate the motion of the heart and improve upon commercially available tracking software. However, the techniques developed are not robust to occlusions or other interferences and cannot automatically correct a failed tracking estimate for the use in real-time cardiac motion tracking.

2.2: Robotic Assistant Surgery Tracking

Open heart and specifically coronary artery bypass surgeries require surgeons to operate on blood vessels that constantly move and cause interference. Research has been conducted to improve the working conditions for surgeons during operation also by using computer vision techniques. The master-slave robotic systems, such as the DaVinci machine, have been improved to include mechanical motion synchronization of the surgical instruments with the beating heart to track, predict, and cancel out the cardiac motion. Active Relative Motion Cancelling (ARMC) is one method used to actively cancel out heart motion by tracking a point of interest on the heart surface to provide surgeons with a still view (Bader et. al, 2007). Three different research studies on the development of cardiac motion tracking systems for this application and their motion tracking technologies will be reviewed.

In a study conducted by Bader et. al (2007), a model-based approach was used to build an estimator to reconstruct multiple feature points from one image frame to the next, to predict cardiac surface motion. A testing model was developed with a circular pulsating membrane, paced between 0.5 and 2.4 hertz, and physical markers to track motion using a stereo camera

system. The motion behavior of the membrane was derived as a system of coupled linear partial differential equations, and converted into a lumped parameter system to estimate the solution of the model at a discrete time point. A Kalman filter was utilized to estimate the state of the lumped parameter system from the location of marked feature points on the membrane model. The location of these points was found through feature extraction from both cameras using an edge detection technique, specifically Canny edge detection (Canny, 1986). The predicted deflection of a series of points that did not include the labeled markers was compared to the actual measured deflection of the labeled points as the model was pulsed at a frequency of 0.653 hertz. The results show that the average prediction error (measured – prediction) was found to be equal to 1.39 millimeters (Bader et. al, 2007).

The model developed from this study demonstrates that the deflection of a circular membrane, which resembles cardiac motion in the z-axis, can be tracked through feature detection. This model also successfully tracked the changes in motion throughout time using non-invasive imaging data. It is reported that at some image frames the Canny edge features could not be found, which contributes to an increase in the prediction error. More research on the types of features to be selected, such as corners instead of edges, could provide a better tracking model to reduce loss from occlusions or other interferences. The membrane model should be tested in real-time on cardiac images to improve non-rigid tracking and be more applicable to robotic-assisted surgery applications.

In a research study conducted by Tuna et al. (2013), two least-square prediction algorithms were created and tested to predict future position estimates of points of interest (POI) from the heart surface. In this study, the heart motion displacement of three calves was recorded using a sonomicrometer system, and imaging data was tested with the algorithms developed. Two piezoelectric crystals were placed near the coronary artery and along the side of the heart to measure displacement. A “one step motion” estimation algorithm was first developed to predict the current POI using the prior position in previous frames through a process of adaptive filtering. Adaptive filtering adjusts the filter weights using a least squares method to allow the algorithm to constantly update based upon the most recent iteration. The second approach developed uses a generalized linear predictor to independently estimate each point over the entire image. This second approach computes an estimation matrix at each iteration, which was found

to be approximately constant throughout all frames. Both algorithms were tested on the pre-recorded cardiac displacement data to track the heart over all image frames. The results of this study show that the generalized predictor root mean square position error was found to be much smaller than the one step estimation at a constant and varying heart rate (Tuna et. al, 2013).

The algorithms developed in this study show that a motion predictor can be developed to actively track specific regions of the heart even when the heart begins to contract at a higher rate. The motion predictors were successfully able to track the non-rigid motion of the heart using non-invasive cardiac imaging data. The predictors were, however, not developed to account for interferences or occlusions, which occur during surgery. It is important to note that even though the prediction algorithms can actively track regions when the heart rate increases, the algorithms cannot track these regions when an abrupt change in heart rate occurs. This abrupt change could be caused from an abnormal arrhythmia or myocardial infarction. The methods presented in this study are also not implemented in real time in an active surgical system.

In a third study, Richa et al. (2010) presented a robust method for estimating three-dimensional temporal and spatial deformation of the heart surface using stereo endoscopic images and computer vision techniques. This method, based on previous research with Thin-Plate Splines (TPS) models, was improved to accurately track large ROI on the heart surface even in the presence of occlusions or tracking failures. In their previous studies, the TPS method was developed to select control points from a reference image of the heart and utilize a warp function to map the control points and the ROI from one frame to the next. It is important to note that the control points are selected manually by the user in a reference image from regions of high texture, such as edges. This study was conducted to improve upon this method by adding the heart motion dynamics to support the tracking accuracy of the TPS solution. A heart motion model was developed based on time-varying Fourier series that is recursively estimated using an Extended Kalman Filter (EKF). This model will be used to reestablish the motion of the heart when a tracking failure occurs. The current algorithm from this study was re-organized to evaluate the quality of the tracking result at every iteration. In the evaluation step, the image alignment error and the estimated three-dimensional heart shape will be checked. If the error is found to be high, the EKF and Fourier series model will be used to restore the tracking of the selected ROI. The alignment error is evaluated by calculating the normalized cross-correlation

coefficient of regions of about 40x40 pixels surrounding each control point. If three of these points were found to be below a set threshold, the current tracking TPS model stops and continues when the control points are visible. The three-dimensional shape analysis is evaluated by calculating the bending energy of the ROI. If the bending energy of any point within this region is greater than a specific threshold, the tracking is stopped and the motion is reestablished from the EKF model. The improved tracking algorithm was tested on human data that was pre-recorded from previous research studies (Richa et. al, 2011).

The algorithm from this study was tested on image sequence from an endoscopic coronary bypass surgery using the DaVinci surgical platform on a human subject. The data consisted of 32 seconds of colored imaging data with a total of 1600 images. Eight control points were selected on the reference image, the EKF was initialized, and the alignment error and bending energy thresholds were set at 0.60 and 0.14 respectively. The researchers note that the first tracking error within the ROI occurred around 3.16 seconds, and the previous method was not able to recover. The new method fixed the tracking error by 3.18 seconds using the predicted heart motion from the EKF and continued. Throughout the whole test, the tracking was suspended for 13.69% of the total time duration. This lead to a computational delay of 800 ms total from tracking loss to reestablishment. The large duration of the delay was found to occur due to the poor prediction quality from abrupt cardiac frequency changes, but the region was successfully tracked throughout the entire sequence (Richa et. al, 2011).

The results show that the non-rigid motion can be tracked through EKF and Fourier series modeling for applications in real time robotic-assisted surgery. The authors of this study suggest that abrupt cardiac frequency changes can be accounted for by incorporating electrocardiogram (ECG) data. The ECG could be utilized to predict abrupt changes in the heart frequency to improve tracking quality (Richa et. al, 2011). However, more instrumentation must be added to the existing method to measure the ECG. The algorithm presented in this study demonstrated that the visual tracking system must be robust to occlusions for robotic-assisted surgery applications. A tracking validation step can also be developed using the normalized cross-correlation coefficient for feature tracking in cardiac motion data.

2.3: Computer Vision Tracking Algorithms for Non-Rigid Motion Tracking

Most of the computer vision techniques initially developed for object tracking and motion detection had an implicit assumption that the object to be tracked in an image was rigid. For biological systems, such as the heart, a tracking system must be able to accurately track and predict the stretching or dilating motion of non-rigid structures, such as a blood vessel. It must estimate these areas of non-uniform motion within one region, which is typically found on the heart. There has not been substantial development of closed-form tracking algorithms for non-rigid heart motion in the literature. However, this problem could be solved through a linearized, piecewise approach. Piecewise motion tracking is used to break up the motion field of a region into smaller areas, called neighborhoods, that can be estimated individually. The next set of studies will present techniques that can be used to model and estimate regions in a scene or image that have different motion fields.

In a study conducted by Cremers and Soatto (2004), a motion estimation and segmentation technique was developed to track multiple regions of interest (ROI) and segment out objects from their respective motion field. This proposed method was developed using Bayesian inference that is updated based on continuous optical flow measures outlined by a contour representation of the motion of separate regions. The motion field is also optimized by a gradient descent minimization factor. The initial image frame is segmented into multiple regions of piecewise parametric motion, where the motion of each region is solved from a system of partial differential equations. With the equations and algorithm developed, this method was first tested on a set of synthetic gray-scale images of similar intensity where the ground-truth motion is known. The results of these tests show that the objects located within the image set were found to be accurately segmented based on their separate motion and not appearance. In a more applicable test, a traffic scene test set was developed by setting an image of two cars to move toward the top-right of the image, while the background of this same image was moved to the bottom-left to simulate camera movement. The motion segmentation method was compared to a previously developed intensity-based segmentation technique for this image sequence. The results show that the intensity-segmentation was only able to segment out the bright and dark areas within the image. The proposed method was accurately able to segment out each individual car from the background, and obtain an accurate estimate of the motion of the cars and background when compared to the ground-truth. A third experiment from this study takes a

traffic sequence of multiple moving cars and a moving background, and estimates the motion of a selected initial segment within the first image. It is important to note that in this test sequence, the ground-truth motion was not known. The results of this test show that the segmentation of one car in the scene was robust to competing motion of other cars and the background. The algorithm began to fail as the car started to move perpendicular to the viewing plane of the image. At this point the optical flow cannot be solved for, and this error is known as the aperture problem. The results of this study show that regions of separate motion can be estimated and the boundaries of these regions can be highlighted to segment out each moving object in a scene. The segmentation method was developed for piecewise motion fields based on parametric motion models. To segment regions of the moving heart, this model cannot be used. However, this technique could be utilized if the specific parametric motion from pre-segmented heart data is learned and used as an input to the motion segmentation algorithm (Cremers & Soatto, 2004).

In another approach proposed by Zhou, Yuan, and Shi (2008), an object tracking technique was developed by combining the use of a scale invariant feature transform (SIFT) with a mean shift algorithm. This algorithm integrated two commonly used object tracking techniques to improve consistency in tracking performance, even if one of them were to fail. A SIFT feature detector is a method that locates points of interest that occur at the maxima and minima of a difference of Gaussian function, across all scale space. Local key points are identified through this step, and then a feature descriptor is developed for each key point to assign measures for robustness of the point against rotation or brightness changes. The final SIFT feature points are chosen by testing the robustness of each key point detected by building an image pyramid and re-sampling the points at each level of the image scale space. The mean shift algorithm works by conducting a color or intensity similarity search using color histograms across two image frames. An initial target window, with its position, is given in the current frame, and the algorithm begins to step through the next image frame in search for a confidence region that has a similar histogram distribution. Both tracking techniques are used in this algorithm by taking measurements and developing an expectation-maximization scheme to achieve a maximum likelihood estimation of similar regions across multiple image frames. A region of interest, or rectangle, is defined in the first frame, along with the computation of the color histogram of this region and the SIFT features. Then in the second frame, the algorithm will examine surrounding areas of the initial position from the previous frame for color similarity measures along with the

sum of square difference between SIFT features. The expectation-maximization will then be used to locate regions that are similar while minimizing the distance between the detected locations from the mean shift and SIFT results. This process will then iterate until the difference between the two is smaller than a set threshold, and the target location will be found in the current frame. The algorithm was tested on four publicly available data sets, and then compared to the SIFT and mean shift detectors separately. The Euclidian distance between the object detection of the three techniques and the ground truth were then compared, and it was found that the mean-shift combination algorithm had a significant lower tracking error for both a single object in a dark scene and tracking a single object in a crowded scene. The combination of both techniques did, however, increase computation time when compared to running each individually. Overall, the proposed algorithm shows promise in improving object tracking over multiple image frames in different scenarios by combining different object tracking techniques together to improve results (Zhou, Yuan, & Shi, 2008).

The combination of different tracking techniques shows an improvement in estimating the motion of one segment in a scene of competing motion. Even though the initial region of interest was assumed to be rigid in all the algorithms presented, this individual consistent tracking can be used for piecewise motion estimation by breaking up a non-rigid region into smaller areas that can be assumed to move as rigid segments.

A third study, proposed by Ren in 2008, improved optical flow results by developing an image-based grouping approach in motion estimation. This method begins by computing a soft edge and texture boundary map using a probability-of-boundary operator, which combines local brightness, color, and texture contrast and differs from traditional edge detection techniques. The boundary map will then be used to develop pairwise affinities between subsequent pairs of points through an intervening contour method. The affinity value represents if two points are separated by strong boundaries or if they belong in a uniform region. The image will then be sampled at corner or edge points for the affinity calculation step, and then the flow will be estimated at these sample points. The affinity values calculated will define a support for the spatial integration of flow to avoid connections across object boundaries, also known as a semi-local flow approach. An affinity-based optical flow will then be calculated across the image to estimate the motion of these points from one frame to the next. The grouping method was then tested on publicly

available data sets and the average angular error and average end-point error was compared with a top-ranking flow estimator from Black and Anandan (1996). The results show that the grouping approach significantly improves the optical flow estimation by lowering both errors measured. These results can be contributed to the improved flow estimation for points that are close to boundaries in an image, but do not necessarily move in the same direction as the boundary. While traditional flow estimators tend to group unwanted regions into areas of strong intensity, like edges and corners, even if the true motion of these points is in a different direction. The downfall to this approach is that the computation time increases due to the increase in the number of points used in computation of the flow field. These results show that grouping optical flow measures by edges or corners improves tracking estimation. This grouping approach can be useful for piecewise estimation as well to estimate regions that undergo non-rigid motion similar to cardiac motion (Ren, 2008).

In summary, existing heart motion tracking systems from imaging modalities were found to be accurate only at specific locations on or within the heart. The systems reviewed here require tagged MRI data, a prior model, or a lot of prior imaging data that can be expensive to capture and have long computation times. Non-invasive optical imaging tracking techniques could improve upon MRI tracking systems presented because optical techniques require less exposure time, are less-invasive, and are more cost effective. The data used to develop this algorithm stems from optical imaging techniques, but could also be used in other imaging modalities. Tracking systems used on robotic surgery applications were found to be accurate. However, these algorithms require the use of artificial sensors and computationally expensive models to effectively track the motion of the heart. Lastly, a set of computer vision techniques that were not originally developed for tracking non-rigid biomedical systems could be used to solve for piecewise motion estimation. Overall, there exists a gap between existing systems and the need of a heart motion tracking approach that is continuous, non-parametric, and robust to tracking failures.

CHAPTER 3: METHODS

3.1: Non-Rigid Motion Observation Model

In order to continuously estimate the non-rigid motion of a biological system, such as the heart, a traditional computer vision technique that assumes a rigid body is no longer directly applicable. A piecewise tracking algorithm is proposed here. It will break up a ROI into smaller components, each estimated as a pseudo-rigid segment. Each small component will be estimated as a pseudo-rigid segment because not every point within that segment will follow the same exact motion. The motion of some points will differ slightly, but the distribution of this motion will follow an approximately rigid model. All the segments combined can approximate the non-rigid motion of a ROI as observed in a two-dimensional space (image).

In order to introduce the piecewise algorithm, a body frame and a camera frame are defined in a three-dimensional space. The body frame is a three-dimensional Cartesian coordinate system attached to the target biological system (the heart), with an arbitrarily chosen origin point (*j*). If the heart were a rigid body, another point (*i*) can be defined at a fixed location in the body frame, X_i^b, which does not change over time in a rigid body frame. However, in a non-rigid system, the location of this point becomes a function of time, i.e. $X_i^b(t)$. At a specific time ($t = t_n$), the location of this point would be at $X_i^b(t_n)$. Since the point (*j*) is the origin, the location of this point in the body frame is zero, i.e., $X_j^b = 0$, regardless of time.

The camera frame is another rigid 3D Cartesian coordinate system. At a time t_n, the origin of the camera frame is located at, for example, $X_c^b(t_n)$, which is a location observed in the rigid body frame. The coordinates of the point (*i*) and the origin (*j*) in the camera frame is thus defined by equation 1 and 2 respectively:

$$X_i^c(t_n) = R_b^c(t_n)[X_i^b(t_n) - X_c^b(t_n)] \quad (1)$$

$$X_j^c(t_n) = R_b^c(t_n)[X_j^b(t_n) - X_c^b(t_n)] \quad (2)$$

where R_b^c is the rotational transform of a point from the body frame to the camera frame. Thus, $(X_i^c(t_n))$ represents point *i* observed in the camera frame at time t_n.

3.1.1: Piecewise Approximation

Now let the time stamp step from t_n to t_{n+1}, still in a non-rigid system. If a rigid body had been assumed in a predictive model, it would not agree with the actual observation. The error (ϵ) between the predicted location and truth is described in equation 3, where $\widetilde{X_i^c}\ (t_{n+1})$ stands for the prediction.

$$\epsilon = prediction - truth = \widetilde{X_i^c}\ (t_{n+1}) - X_i^c(t_{n+1}) \tag{3}$$

Both terms can be substituted from equation 1 and simplified below in equation 4:

$$\epsilon = R_b^{C_n+1}\left[X_i^b(t_n) - X_c^b(t_{n+1})\right] - R_b^{C_n+1}\left[X_i^b(t_{n+1}) - X_c^b(t_{n+1})\right] \tag{4}$$

As afore mentioned, in a non-rigid system, $X_i^b(t_{n+1}) \neq X_i^b(t_n)$. We further assume that the location of point (i) has been biased through deformation by Δb, such that

$$X_i^b(t_{n+1}) = X_i^b(t_n) + \Delta b$$

The prediction error is therefore also a function of Δb:

$$\epsilon = R_b^{C_n+1}\left[X_i^b(t_n)\right] - R_b^{C_n+1}\left[X_i^b(t_n) + \Delta b\right]$$

where $R_b^{C_n+1}$ is the rotation transform from the body frame to the current camera frame. We can then simplify the equation above into:

$$\epsilon = -R_b^{C_n+1}\Delta b \tag{5}$$

The difference between the rigid and the non-rigid motion is now observed in the camera frame, as shown in equation 5. In a non-rigid system, the bias Δb can be defined as a function of time and space shown in equation 6 below:

$$\Delta b(t_{n+1} - t_n, \overrightarrow{X_i^b - X_j^b}) \tag{6}$$

In a piecewise approach, the non-rigid motion will be approximated with pseudo-rigid motion in a small neighborhood, which is observed as a small "patch" in this image. In order for this approximation to be sufficiently accurate, a constraint must be applied to Δb such that it can be linearized. In general, the bias term is assumed to be continuous over time and space in a biological heart. The bias term must fit the continuity constraint as described below:

$$\Delta b\left(t_{n+1}-t_n, \overrightarrow{X_i^b - X_j^b}\right) \text{ is smooth if } \frac{\partial \Delta b}{\partial t}, \frac{\partial \Delta b}{\partial X} \text{ exist and are continuous}$$

For this function to be smooth, it must be differentiable in time and space everywhere in its domain. Also, all the partial derivatives must exist and be continuous in all orders across the domain. For the function to be continuous, it must be defined at any point, and the value at that point must equal a real number. The limit of the function must also exist and be equal to the function value at that point.

Δb is a function of time and space because its value will change overtime, or consecutive image frames, and at different points in the ROI. The bias within a small patch can be approximated using first order Taylor series and is shown below in equation 7:

$$\Delta b\left(t_{n+1}, X_i^b\right) \cong \Delta b\left(t_n, X_j^b\right) + \frac{\partial b}{\partial t}(\Delta t) + \frac{\partial b}{\partial x^b}(\Delta x^b) + \frac{\partial b}{\partial y^b}(\Delta y^b) + \frac{\partial b}{\partial z^b}(\Delta z^b) \qquad (7)$$

where Δt is the change in time from t_n to t_{n+1} and $(\Delta x^b, \Delta y^b, \Delta z^b)$ represents changes along the three axes in the body frame. To further simply the analysis, in equation 8, we assume that the bias is only sensitive to the distance between two points, such that

$$\Delta b\left(t_{n+1}, X_i^b\right) \cong \Delta b\left(t_n, X_j^b\right) + \frac{\partial b}{\partial t}(\Delta t) + \frac{\partial b}{\partial D}(\Delta D) \qquad (8)$$

where ΔD is the change in distance from a point (*i*) to the origin (*j*) within one patch.

Since the bias is assumed to be continuous and smooth, if the change in time between image frames is approximately small ($\Delta t \cong 0$), the bias due to the time change will be sufficiently small. The bias within a patch due to distance can then be defined by evaluating equation 8 using the distance variable only:

$$\Delta b\left(t_{n+1}, X_i^b\right) \cong \Delta b\left(t_n, X_j^b\right) + \frac{\partial b}{\partial D}(\Delta D)$$

Similarly, there would exist a small neighborhood surround the origin (*j*), in which any point (*i*) has $\left|\frac{\partial b}{\partial D}\right| \cong \mathbf{0}$. In other words, there is a pseudo-rigid relationship between both points. This neighborhood, will be called a "patch" in this work. If a patch was observed to be approximately rigid at a time (t_n), it still will be approximately rigid at the next time step (t_{n+1}). Therefore, the bias change due to time will be minimal, i.e., $\Delta b\left(t_{n+1}, X_i^b\right) \cong \Delta b\left(t_n, X_i^b\right)$.

Piecewise motion estimation also requires a constraint across neighboring patches. Neighboring patches do not necessarily share the same linear transformation. However, their motion models are expected be continuous. Assume that a point (k) is located in a patch adjacent to the one defined with origin (j). The bias term at point (k) can be approximated with a first order Taylor series following equation 9 below:

$$\Delta b\left(t_{n+1}, X_k^b\right) \cong \Delta b\left(t_{n+1}, X_j^b\right) + \frac{\partial b}{\partial D}(\Delta D) \tag{9}$$

The variation over time is assumed to be negligible. Both patches will follow a pseudo-rigid model. Since Δb is smooth, the first order Taylor series is sufficient to estimate the bias if the distance between both points ($\Delta D = \left|X_k^b - X_j^b\right|$) is small enough. It provides yet another constraint on the patch size. With smaller patches, the bias Δb between neighboring patches would have a limited difference shown in equations 10 and 11 below.

$$\left|\Delta b\left(t_{n+1}, X_k^b\right) - \Delta b\left(t_{n+1}, X_j^b\right)\right| < limit \tag{10}$$

In other words,

$$\left|\left(X_k^b(t_{n+1}) - X_k^b(t_n)\right) - \left(X_j^b(t_{n+1}) - X_j^b(t_n)\right)\right| < limit \tag{11}$$

3.1.2: 2D Observations and Constraints

The non-rigid motion of point (i), is observed in a two-dimensional space (image). First, the location of this point can be converted from body frame to camera frame, as afore mentioned:

$$X_i^c(t_{n+1}) = R_b^c(t_{n+1})[X_i^b(t_{n+1}) - X_c^b(t_{n+1})]$$

$X_i^c(t_{n+1})$ is defined in the three-dimensional camera frame, with x, y, and z components: $X_i^c(t_{n+1}) = [x, y, z]$. Conventionally, x points to the right, y points down and z points out of the camera lens. The location of (i) will be observed in an image that has normalized x and y components, which will then be compared to the predicted location of this point as shown below (Hartley & Zisserman, 2004):

$$Seen: X_i^{2D}(t_{n+1}) = \left(\frac{x}{z}, \frac{y}{z}\right)$$

$$Prediction: \widehat{X_i^{2D}}(t_{n+1}) = \left(\frac{\hat{x}}{\hat{z}}, \frac{\hat{y}}{\hat{z}}\right)$$

The prediction, once again, is assumed to follow an approximately rigid model. If the observed location in (x, y, z) is approximately equal to the prediction in ($\hat{x}, \hat{y}, \hat{z}$), i.e. $\Delta b \sim 0$, then the prediction is accurate and the tracking error (ϵ) could be minimized. It is important to note that the z component represents the distance from the camera frame to the body frame. In a medical imaging system, the distance between the camera and target is unlikely to dramatically change. Therefore, z is assumed to be approximately equal to the predicted $\hat{z}$ in both locations ($z \cong \hat{z}$). We assume that the majority of motion will be observed in the x and y directions.

Constraint 1: Within a patch that is sufficiently small, the pseudo-rigid prediction is expected to agree with observation

$$\epsilon = prediction - truth \cong 0$$

$$\widehat{X_i^{2D}}(t_{n+1}) \cong X_i^{2D}(t_{n+1})$$

Any disagreement would be attributed to noise in feature tracking, which is expected to follow Gaussian distribution. As to be discussed in the following sections, motion of a pseudo-rigid patch can be modeled by a linear transform, such as the fundamental matrix or a homography matrix. It allows us to predict the motion of any single point based on the consistency of neighboring points. Furthermore, since we assume that the 3D motion between neighboring patches would have a limited level of difference, the difference in 2D motion would also be limited.

Constraint 2: When the difference between the neighboring patches is sufficiently small,

$$\left|\left(\widehat{X_k^{2D}}(t_{n+1}) - \widehat{X_k^{2D}}(t_n)\right) - \left(\widehat{X_J^{2D}}(t_{n+1}) - \widehat{X_J^{2D}}(t_n)\right)\right| < limit$$

It is realized that the difference is unknown and will differ over location and time. However, when observed over a large number of points over time, it is assumed to follow a Gaussian distribution between two patches. The quality of non-rigid motion tracking can be quantified by how well the tracking residuals can meet this criterion.

3.1.3: Configuration of Piecewise Tracking

The first constraint assumes that the motion within a patch is approximately rigid, and in that case, the tracking residuals will be dominated by noise. We assume that with the right patch

size, these residuals will follow a normal distribution with the number of outliers no greater than two sigma or be greater than five percent of the motion within a patch in the x and y directions. This pseudo-rigid motion can then be estimated with all the point features observed in this patch, through a computer vision technique such as a homography or averaged optical flow estimation.

The second constraint assumes that neighboring patches are interconnected, and that the motion from patch to patch is continuous. The second threshold will be determined by sampling the motion, both in the x and y directions, at every patch through a set number of image frames. The motion differences between neighboring patches in the same row and column will also be calculated. The motion across neighboring patches will also be assumed to follow a normal distribution. This assumption will be confirmed through a chi-square goodness of fit test of the motion differences between neighboring patches. The motion differences at every patch with its corresponding neighbors across a set number of image frames will be collected to estimate the mean and standard deviation. The mean plus two sigma, or 95th percentile point, will be calculated for every distribution, and the maximum value among all distributions will then be used as a threshold. Four difference thresholds will be calculated in total. The first two are the horizontal and vertical differences for the motion in the x-direction, and the next two as the horizontal and vertical differences in the y-direction. Each motion difference will be compared against the corresponding threshold, and will be flagged if it exceeds the threshold.

For every individual patch, both constraints are checked at every frame to quantify the quality of motion prediction. If either constraint is not met, the tracking estimate is potentially incorrect and will be flagged. If the constraints are still not met two more times in a row, it is considered a tracking failure, and reinitialization step must be completed among the discontinued image frames to restart the piecewise tracking process.

The algorithm has been illustrated in a flowchart, as shown in Figure 1 below. The flowchart presents an outline of the necessary steps to track an input region selected from the user, and estimate the motion at every image frame. The algorithm will begin by preprocessing the image data set through histogram equalization to improve the overall contrast of the image. The user will then be asked to input the starting and stopping point to define a region of interest to be tracked. Next, the appropriate size, number of patches, and the motion difference thresholds will be determined. The appropriate edges and detected corner features will be then stored in

each patch from frame one, in the next step, to be used as the initial frame in tracking. A Kanade-Lucas-Tomasi (KLT) optical flow approach will then be used to move corner features from the previous frame to the current frame to be used as an optical flow measure for the ROI (Shi & Tomasi, 1994). The next major component of this algorithm will be to check the quality of tracking by determining if the flow within a patch is normally distributed, and check if the average flow measure between neighboring patches is continuous. If the quality of tracking passes this set of criteria, the edges from the previous frame will then be moved by the average optical flow measure of each patch. If the quality of tracking was found to fail either set of criteria, this frame will be flagged. If a continuous failure is found for up to three flags, the reinitialization step will follow. For the reinitialization step, a homography transform will be used to realign the piecewise grid and relocate the features to be tracked. Highlighting the edge points moved to the next image frame will also provide qualitative results to determine if the non-rigid region is accurately tracked as pseudo-rigid patch segments. The algorithms used in each component of this flowchart will be presented in detail in the following section.

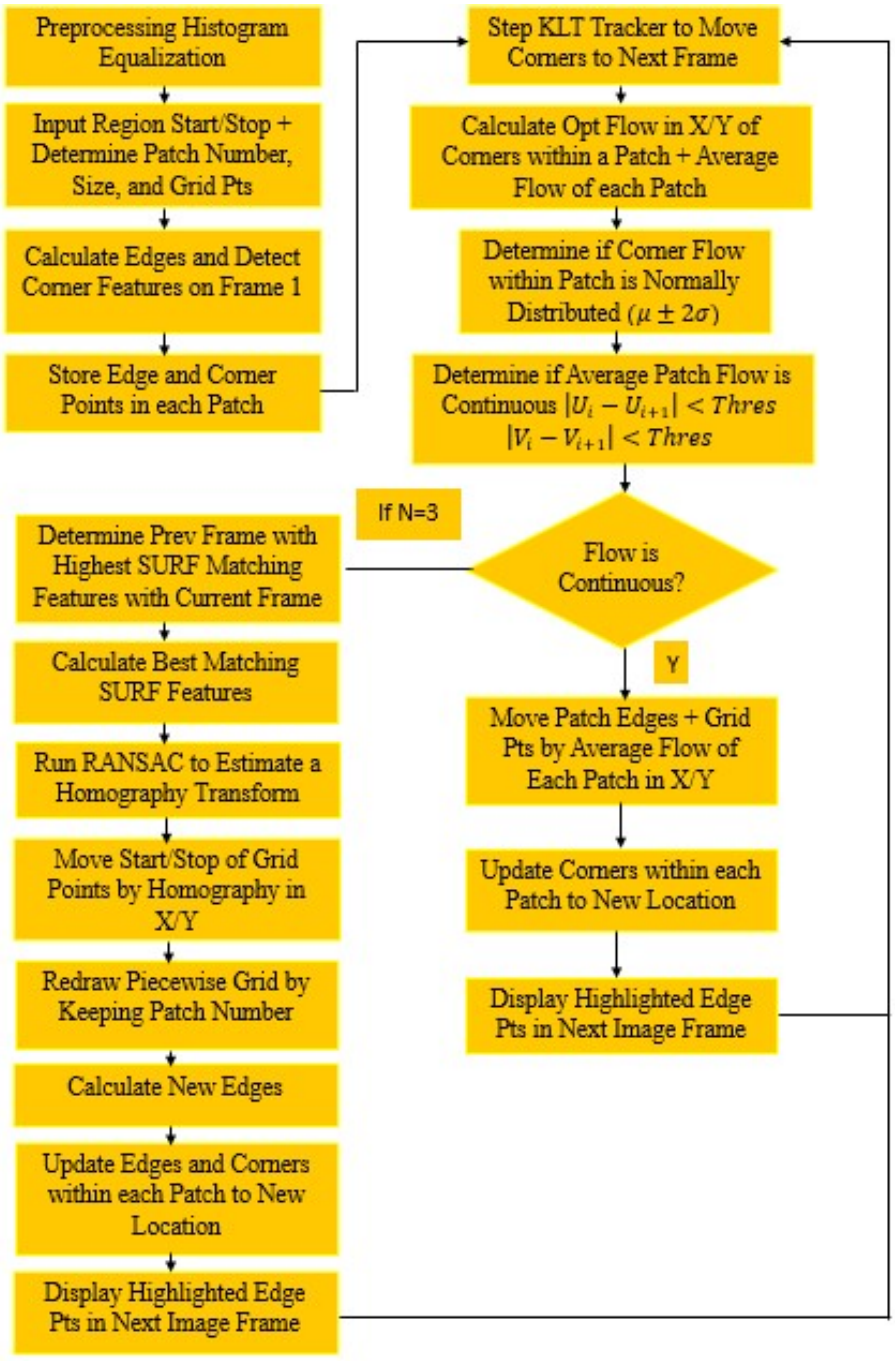


Figure 1: Algorithm Flowchart

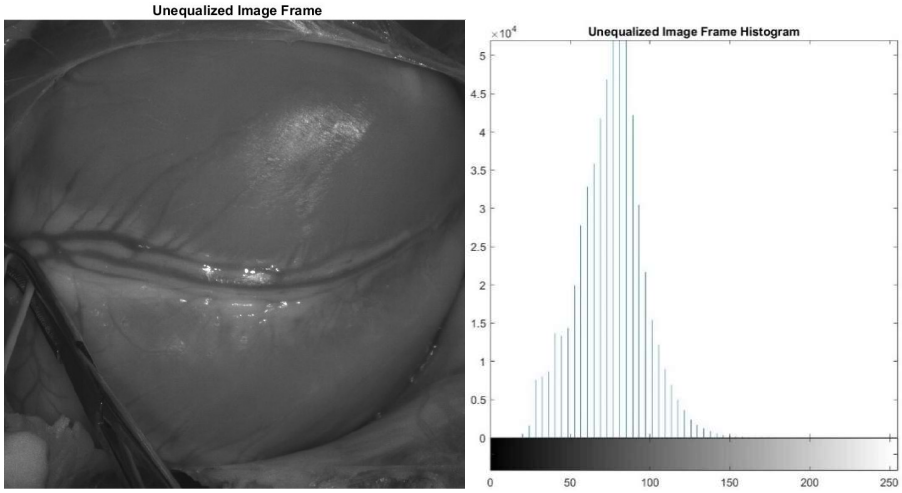
3.2: Computer Vision Techniques

The algorithm has been developed using multiple computer vision techniques and has been implemented in MATLAB. The following sub-sections will introduce the theory, implementation, and simulated results to show how each technique works in relation to the

overall process. Section 3.2.1 will present the preprocessing equalization, edge detection, and calculation to determine the appropriate patch number and size. Section 3.2.2 will present the corner detection step, motion estimation or optical flow technique, and the quality of flow estimation steps. Section 3.2.3 will define the use of Speeded Up Robust Features (SURF) in determining matching feature points between two discontinued image frames. This section will then continue to outline the Random Sample Consensus (RANSAC) process using these SURF feature points to estimate a homography transform to reinitialize the piecewise tracking process.

3.2.1: Preprocessing: Histogram Equalization, Determine Correct Patch Size, Edge Detection

The initial data from the imaging system must be preprocessed to improve the overall visual quality of the images acquired. A histogram equalization function will be applied on every image as the first component of the algorithm. This function will first evaluate the intensity distribution, or histogram, from one or a few images. The normalization process maps a given distribution to another wider and uniform distribution so that the intensity values are spaced out over the entire range (MathWorks, 2017). This MATLAB function works by taking the cumulative distribution function of the image intensity values and performing a transform to develop a linearized cumulative distribution function of the original image. The linearized distribution can be mapped back into an image that will have improved contrast and a spaced-out histogram. Figure 2 below demonstrates the histogram equalization process for two images of one of the heart data sets.



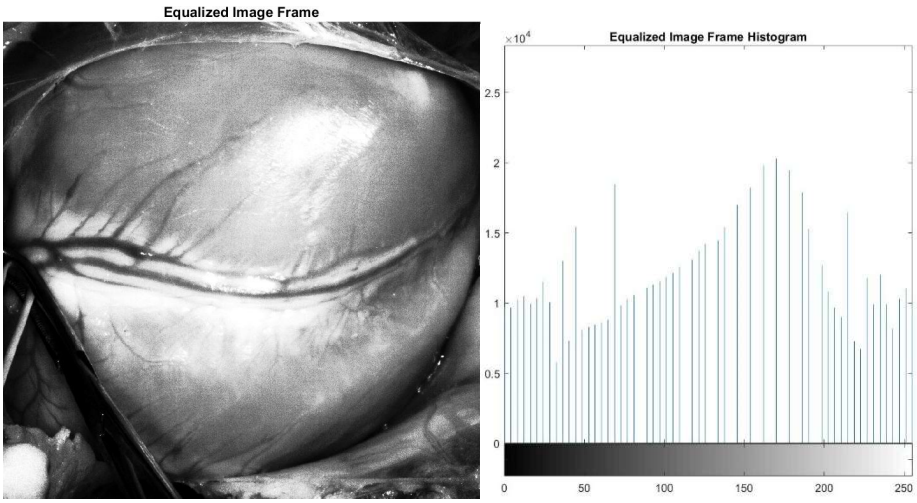


Figure 2: Histogram Equalization Results – Original Image and Respective Histogram (top) Equalized Image and Histogram (bottom)

The next step after histogram equalization is to determine the correct number of patches and the size of each at the beginning of the tracking process. To determine the correct number of patches, a separate function was developed to estimate the quality of the motion between the first two frames by searching for the optimal setting for the number of patches. The motion estimation technique is based on corner feature tracking within each patch, and will be described in the next section. Each patch will be evaluated on the bias against both constraints: the motion within a patch must be assumed to be rigid, and the motion across neighboring patches must be continuous. To fit the first constraint, the motion within each patch will be tested for normal distribution through a chi-square goodness of fit and outlier detection tests. The motion between neighboring patches will also be tested to fit a normal distribution in the horizontal and vertical directions by using a chi-square goodness of fit test. A maximum threshold of each neighboring difference will then be calculated to represent the maximum offset allowed for motion among neighboring patches in either the x or y directions, as pointed out in constraint two of the piecewise tracking approach.

Edge detection is a key component in the tracking algorithm because locating the boundaries found within an image can provide the most useful information to identify a ROI. This technique will be used also as a preprocessing step to define the initial boundary points of the ROI within each patch. An edge can be defined as a transition point of the gray or pixel level of the image as it changes from an area of low values to high values or vice versa (Phillips,

2000). The edges of an image must first be detected to determine the main components of the image and provide information from the image. An image array mask is combined with the original image array data to detect and highlight the edges through a process called convolution. The output array data from this convolution will result in a reduction in overall noise of the image, and show the outline of the objects represented in the image. A correlation kernel, also known as a mask, can be convoluted with the original image pixel matrix data. The convolution operation can be shown in the demo provided by Figure 3 (Gimp, 2017). The resulting output image is the weighted sum of neighborhood input pixels from this function.

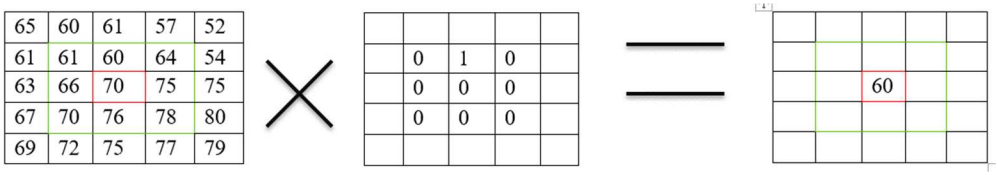


Figure 3: Convolution Demonstration

An image matrix is illustrated on the left side in Figure 3. Each pixel is marked with its intensity value, and the center pixel is outlined in red. The kernel is then applied to the area that has a green border. In the middle is the kernel matrix and, on the right, is the convolution result. The initial pixel intensity (70) has become 60: (61*0) +(60*1) +(64*0) + (66*0) +(70*0) +(75*0) + (70*0) +(76*0) +(78*0) = 60. As a graphical result, the initial pixel moved a pixel downwards. This convolution process is applied to locate the edges using a mask with specified values that can be changed. Figure 4 below shows the original image with the edge detection output images for a low threshold and high threshold. A canny mask is applied to the image using the MATLAB edge function, where the strength of the mask can be tuned by decreasing the threshold parameter (Canny, 1986). Increasing the threshold value, will lead to a decrease in the amount of edges detected. The edge detection algorithm must be tuned based on the application. With the edge locations known, each edge will be moved and highlighted from frame to frame based on optical flow or homography estimation.

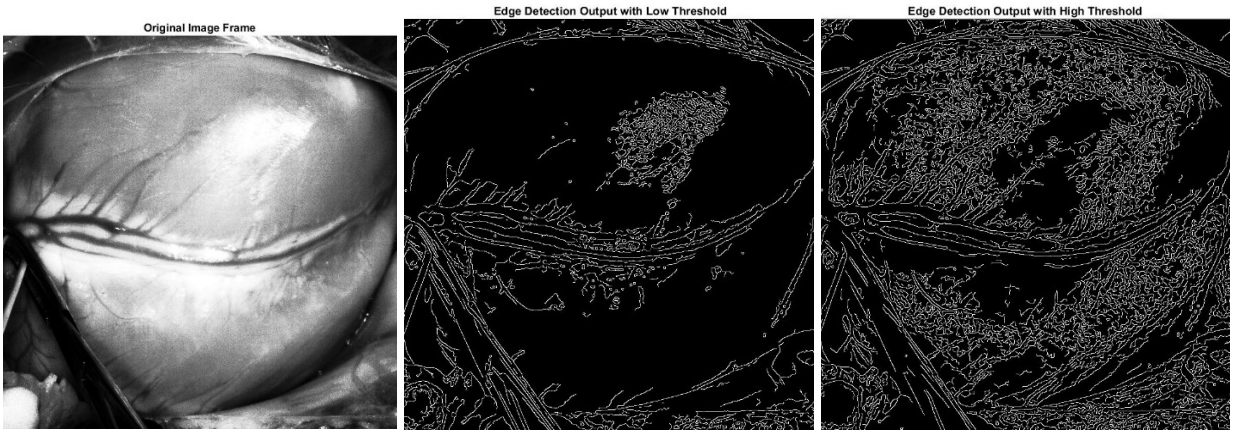


Figure 4: Original Image (left) with Low Threshold Edge Detection (middle) and High Threshold Edge Detection (right)

3.2.2: Corner Detection and Kanade-Lucas-Tomasi Tracking (KLT)

With the image pre-processing and initialization steps completed, the ROI motion can be estimated. There exist multiple methods to estimate an object's motion. A commonly used method is to solve for the optical flow of the object. Optical flow can be defined as the two-dimensional displacement or velocity estimation of pixel patches on an image plane. Figure 5 below shows an example of optical flow between two video frames (t and $t + 1$) with pixel points ($p_{1,}p_2, p_3$). Computing the optical flow of the two frames of this video sequence results in velocity vectors ($v_{1,}v_2, v_3$) to estimate the apparent motion of these points.

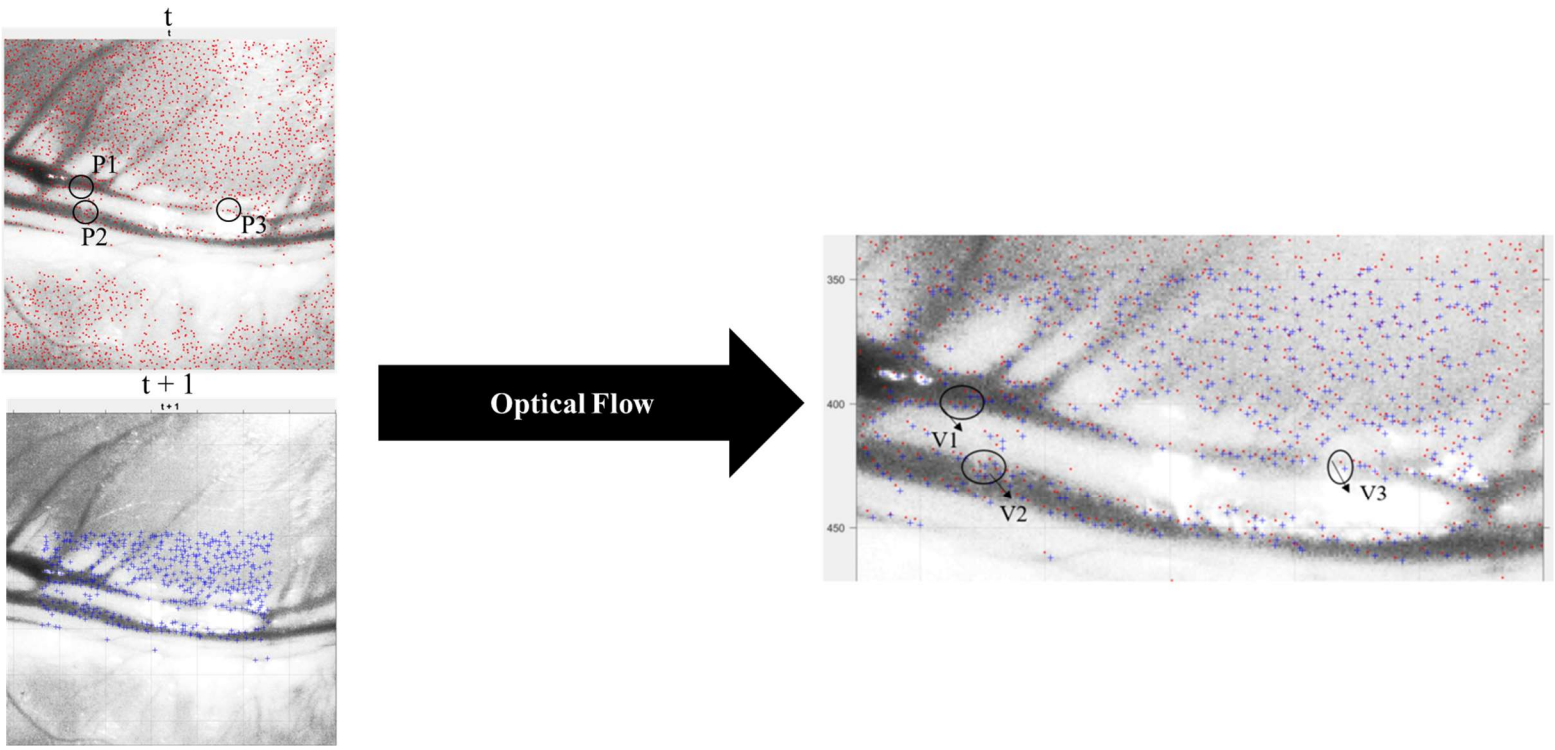


Figure 5: Optical Flow of 2 Image Frames with Points (left) and Velocity Estimation (right)

The common optical flow techniques developed must define some assumptions and constraints to estimate the motion of points within an image. One important assumption is that the brightness (intensity) of a point remains constant from one frame to the next, even though the position of that point changes (Cremers & Wedel, 2011). The brightness constancy constraint is also labeled as the optical flow constraint, and is shown in equation 12 below.

$$I_t + uI_x + vI_y = 0 \tag{12}$$

Where I_x, I_y, I_t are the partial derivatives of the image with respect to x, y, and t. While u and v are the motion vectors in the x and y direction, respectively, that are to be estimated. Figure 6 below will show example images of the image partial derivatives in the x and y directions by applying an image gradient. Another optical flow assumption is spatial coherence, where neighboring points in an image frame typically belong to the same surface and have similar motions between image frames. A third optical flow assumption is temporal persistence, where the motion of an object, or group of points, within an image changes gradually over time. In the case for most optical flow approaches, the apparent motion of the points within an image frame is assumed to be small throughout subsequent frames. Regions within the image for optical flow movement must also avoid "bad" textures that include homogenous intensity values and areas of

linear symmetry. A classic example of the linear symmetry problem is the barber pole illusion shown in Figure 7 below. The true movement of the stripes within the image is horizontal, however the optical flow and perceived motion is that the stripes are moving up along the z-axis. Two additional constraints are typically added to solve this problem. The flow field is assumed to be smooth locally, and the optical flow is solved within a specific size window that is swept over the image to estimate the true motion. The barber pole illusion can then be solved along the outside edge where it is estimated that the information within the image is moving horizontally.

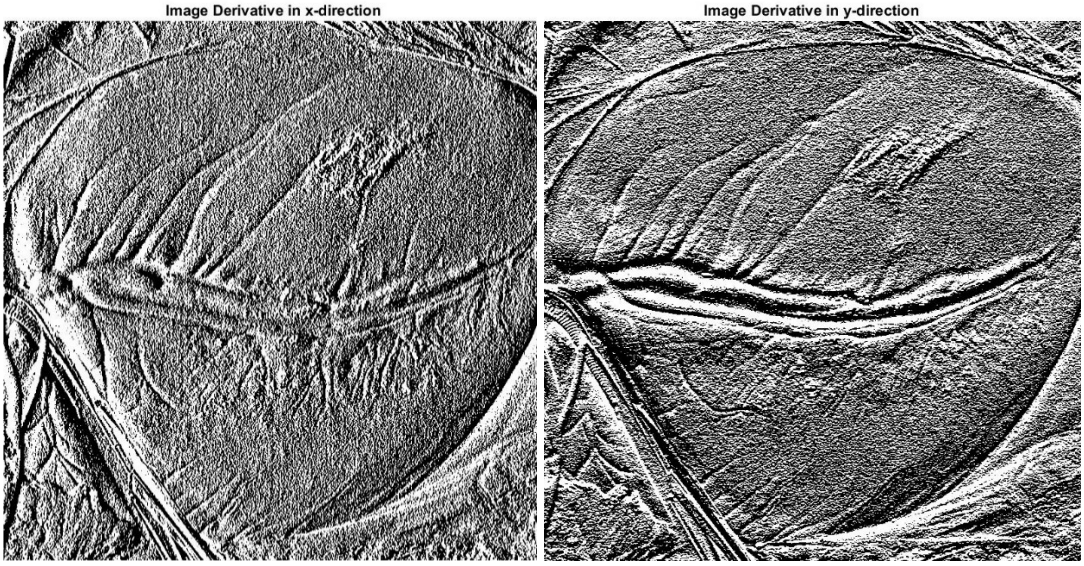


Figure 6: Image Derivatives in the X (left) and Y (right) Directions

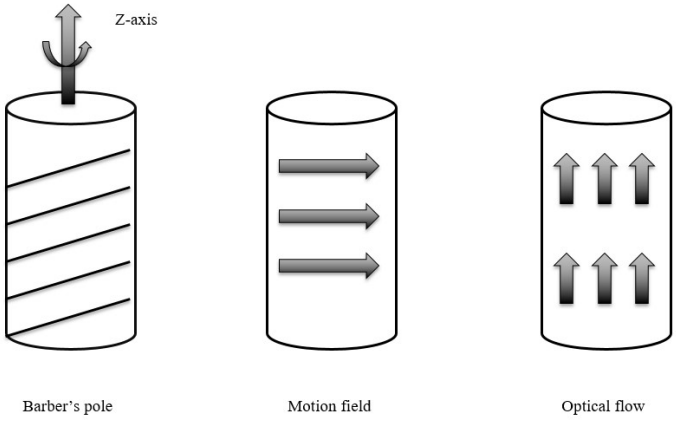


Figure 7: Barber Pole Illusion with True Motion (middle) and Incorrect Optical Flow (right)

There exist many different solutions to the optical flow equation, and a variation of the Lucas-Kanade (LK) method will be used to compute the optical flow in this case (Lucas & Kanade , 1981). The LK method works by assuming motion within a small window is uniform,

typically NxN where N is smaller than 15 pixels. The optical flow constraint is then evaluated at all pixels within the defined neighborhood to estimate their respective motion, and this equation is shown below in equation 13. Equation 13 is then solved by applying a least squares approach to a quadratic equation that is derived below to form equation 14 (Cremers & Wedel, 2011).

$$\min_{u,v}[\Sigma\left(uI_x + vI_y + I_t\right)^2] \tag{13}$$

which gives,

$$\Sigma\left(uI_x + vI_y + I_t\right)I_x = 0 \; and \; \Sigma\left(uI_x + vI_y + I_t\right)I_y = 0$$

$$\Sigma I_x^2 u + \Sigma I_x I_y v = -\Sigma I_x I_t \; and \; \Sigma I_x I_y u + \Sigma I_y^2 v = -\Sigma I_y I_t$$

$$\begin{bmatrix} \Sigma I_x^2 & \Sigma I_x I_y \\ \Sigma I_x I_y & \Sigma I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\Sigma I_x I_t \\ -\Sigma I_y I_t \end{bmatrix} \tag{14}$$

Where Σ represents the summation of all pixels within a specified neighborhood for each term that applies. This partial differential can be solved for all points within the neighborhood, but it is important to note where equation 10 produces the best results. The most accurate flow estimates occur at regions with have high texture or sharp intensity change that are represented as corners, edges, and areas of large gradients. The most inaccurate optical flow measurements occur at areas with low gradients that have small or no intensity change (Cremers & Wedel, 2011). The optical flow results can also be improved through iterative refinement by using image pyramids. The optical flow motion between two points is solved for in low resolution images first, and then refined on increasingly higher resolution images. This pyramid refinement step can be seen in Figure 8 below. Overall, the LK optical flow approach was developed to estimate the movement of rigid structures as the most accurate flow estimations occur at edges or corners. To apply this technique to estimate the non-rigid motion of the heart, the optical flow measurements must be sampled in small patch areas that are assumed to follow rigid motion. Using the LK method does produce accurate results to predict the motion of the heart structure in the first couple of frames. However, the location of the targeted vessel edges tends to drift away from their actual location that cannot be recovered even after employing a re-alignment step. This result shows that the motion estimation is not exact with a neighborhood approach and must be improved.

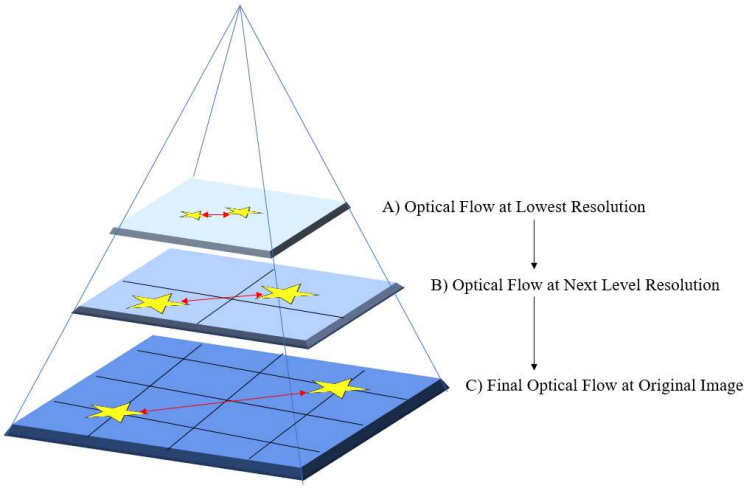


Figure 8: Iterative Refinement of Optical Flow using Image Pyramids

The Kanade-Lucas-Tomasi (KLT) tracker will be used instead of the LK approach to estimate the motion of corner points from frame to frame. The Harris corner features are used in this approach, because they are well-defined as an intersection of two edges and are robust to large motion or intensity changes from one frame to the next (Shi & Tomasi, 1994). The KLT algorithm, developed in 1991, works to improve the optical flow estimation by evaluating regions that propose a well-condition system to solve for the motion of an object or region (Tomasi & Kanade, 1991). This algorithm was improved to its current implementation in 1994 and follows the outline shown below (Shi & Tomasi, 1994):

1. Detect Harris corner features
2. For each Harris corner compute motion between consecutive frames using either the LK method or affine motion estimation
3. Store motion vector of each corner and update corner position in new frame
4. (optional) Add more corner points every 10 frames using step 1
5. Repeat motion estimation steps and constantly update corner position in current frame

The KLT algorithm does produce improved results for estimating the non-rigid motion of the heart structure. The detected corners will be sampled by each patch and moved individually to develop an average motion for every patch. Figure 9 below shows an example of finding the corners in an image frame and showing their moved location in the next frame. The movement of the corners from frame 1 can be seen for this ROI to the right and down into the frame 2 corners.

Estimating the motion with this technique cancels the drifting result of the LK motion estimation and keeps the ROI prediction consistent at every frame.

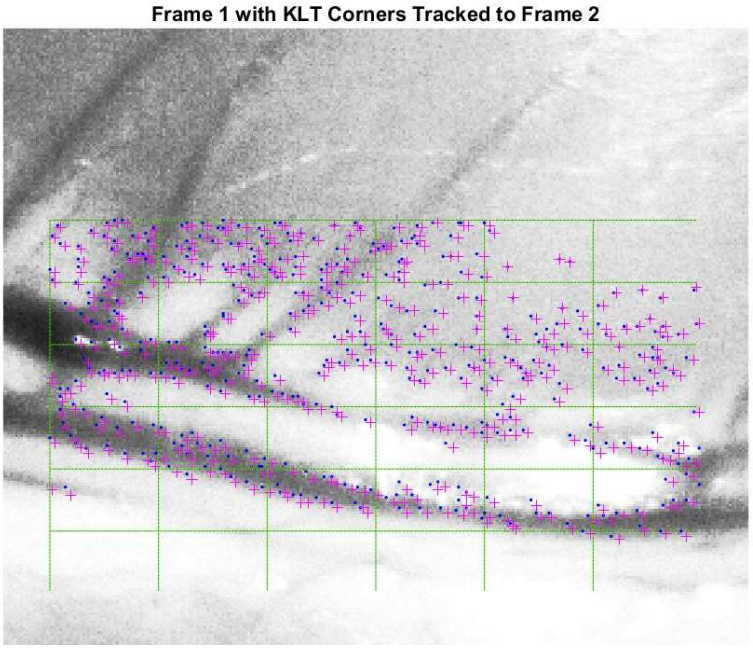


Figure 9: Image 1 ROI with Corners from Frame 1 (Dot) and Frame 2 (Plus)

With the optical flow estimated, the quality of tracking must be checked before the ROI is moved to the next frame. The quality of tracking will be checked by determining if the flow within a patch is normally distributed, and if the average flow measure between neighboring patches is continuous. To begin this process, the optical flow estimation within each patch will be checked for a normal distribution by determining the number of outliers outside of the Gaussian distribution ($\pm 2\sigma$) and a chi-square goodness of fit test. The difference between the average motion of two neighbors within a row will be compared along with the difference between two neighbors within a column to fit a predefined threshold. This will be done to ensure the estimation is accurate and does not break up the continuous motion constraint. If both constraints are met, the optical flow estimation will be used to move each patch into its respective location in the next image frame. If both constraints are not met, the optical flow estimation failed and will be flagged to count the number of failures within a small number of frames. The flag count will continue to iterate as long as the next optical flow estimation does not meet the same constraints. The piecewise ROI will then need to be reinitialized using the SURF/RANSAC technique after a set number of flags is reached. The addition of flagged

estimates will allow the KLT tracker to fix any error in real time before moving on to the reinitialization stage.

3.2.3: Initialization and Reinitialization Techniques

If the motion estimate was found to not pass the quality criteria as normally distributed within a patch and continuous across neighboring patches, the ROI must be reinitialized by determining a homography transform. This transform can be estimated by locating the matching SURF features within an image and use RANSAC to estimate a homography. To begin describing this process, feature matching must be introduced. Feature matching is an important image-processing technique to locate corresponding points of a specific object within two image frames. The Speeded Up Robust Features (SURF) algorithm can be utilized as a scale and rotation-invariant detector and descriptor to approximate matching pixels between two images for object tracking. The algorithm, developed by Bay et al., can be summarized into three steps. First, specific feature points are located at pixels corresponding to high frequency components, typically found in corners, blobs, edges, and T-junctions (Bay et al., 2007). Next, the neighborhood of every feature point detected will be represented by a feature vector, called a descriptor. The descriptor must be unique and robust to noise, displacement, and photometric deformations (Bay et. al, 2007). Lastly, the descriptor vectors of each detected point are then matched between the two images. The matching step is done by determining the minimum Euclidean distance between two features. Each step in the SURF matching process will be described in detail below.

The first objective in the SURF algorithm is to locate interest points that will be used to determine matches between two images. These interest points typically are found in corners, edges, blobs, and T-junctions (Bay et. al, 2007). A Hessian matrix approximation is used in this algorithm with integral images to reduce computation time and improve efficiency. An integral image at a pixel location (x) is represented as the sum of all local pixel intensities within a rectangular region. Equation 15 below shows the mathematical calculation of the integral image:

$$I_{\Sigma}(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \tag{15}$$

Where I is the input image, **x** is the sum of all the pixel intensity values (Viola & Jones, 2001). A graphical representation of an integral image is shown in Figure 10 below. This figure shows the

representation of a rectangular area as three additions to calculate the sum of the intensities. This allows for an increase in processing time for convolution of filters, and, in the case of SURF, allows for a faster interest point detection step.

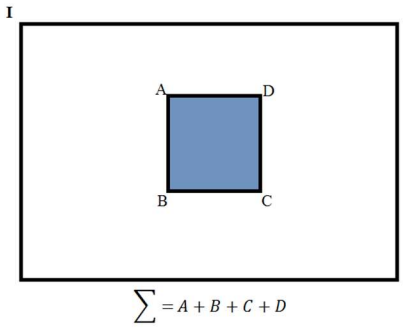


Figure 10: Graphical Representation of an Integral Image.

The SURF feature point detector is based off a Hessian matrix approach to detect blob-like structures at locations where the determinant is maximum. At a given point within an image (x), the Hessian Matrix H(x,σ) is defined in equation 16 below:

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{16}$$

Where Lxx is the convolution of the Gaussian second order derivative with the image I in the point x, similar for Lxy and Lyy (Bay et. al, 2007). Gaussian derivatives are approximated at a low computation cost using box filters and integral images to speed up the convolution step and reduce computation time. The three 9x9 box filters used on each image in the x, y, and both x and y directions are shown in Figure 11 below as Dxx, Dyy, and Dxy respectively (Juan et. al, 2010).

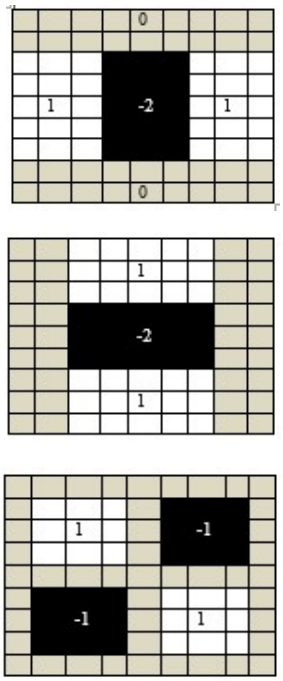


Figure 11: 9x9 Box Filters used in Convolution with Image to Approximate Gaussian

The approximated determinant of the Hessian matrix, shown in equation 17 below, represents the blob response in the image at a specified location x. The results are stored in a blob response map over different scales, where the local maxima are detected.

$$\det\left(H_{approx}\right) = D_{xx}D_{yy} - \left(0.9D_{xy}\right)^2 \tag{17}$$

For the SURF feature point detection step, points of interest must be found at different image scales to locate the best feature points that show up in every down-sampled size of an image. The scale-space can be represented as a pyramid, and is analyzed by up-scaling the box filter size, instead of iteratively reducing the image size shown in Figure 12 below. The output of the 9x9 box filter is set as the initial scale labeled s = 1.2. The following results are calculated by filtering the same image with bigger masks with the use of integral images. The box filter, or mask, size must be increased by 6 pixels at each iteration to keep the filter size uneven and keep the central pixel (Bay et. al, 2007).

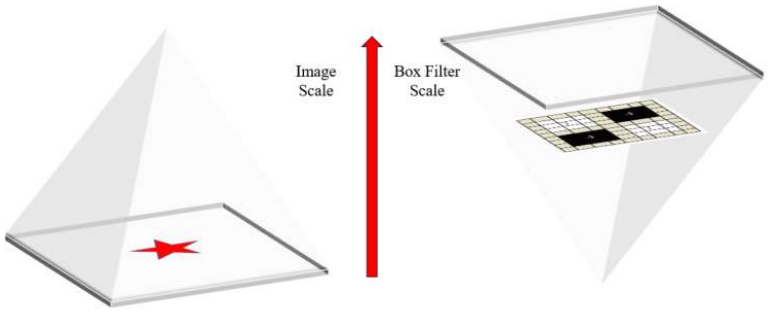


Figure 12: Increase in Scale of the Filter (right) using Integral Images Reduces Computation Time

With multiple interest points detected across different scales, the next step is to localize the points found using a non-maximum suppression in a 3 x 3 x 3 neighborhood. A Fast-Hessian detector method, proposed by Brown and Lowe, is used by interpolating the maximum of the determinant of the Hessian matrix in the scale and image space. The interest points located can be shown in Figure 13 below as an example image from the heart data set with detected interest regions.

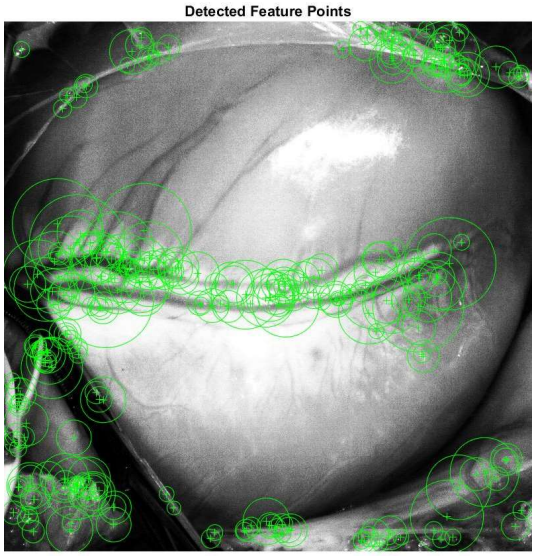


Figure 13: Fast-Hessian Detection of Feature Points in an Image

The next component of the SURF algorithm is the development of a descriptor for each feature point detected. The first step in the descriptor process is to identify a reproducible orientation for each interest point detected. The Haar wavelet responses in the x and y direction

are calculated within a circular neighborhood of the interest point detected, shown in Figure 14 below. The wavelet responses are calculated and weighted with a Gaussian function centered at the interest point. Every sample point response within the circular region of the detected point can be represented as points in a circular space, also shown in Figure 14 below, with horizontal response strength in x and vertical strength in the y. The dominant orientation is estimated by calculating the sum of all sample points within a sliding orientation window of size $\frac{\pi}{3}$. This results in an orientation vector for each point. For many applications, however, rotation invariance is not needed. Upright SURF (U-SURF) can be used instead due to faster computation time while maintaining a robustness to rotation of about ±15° (Bay et. al, 2007).

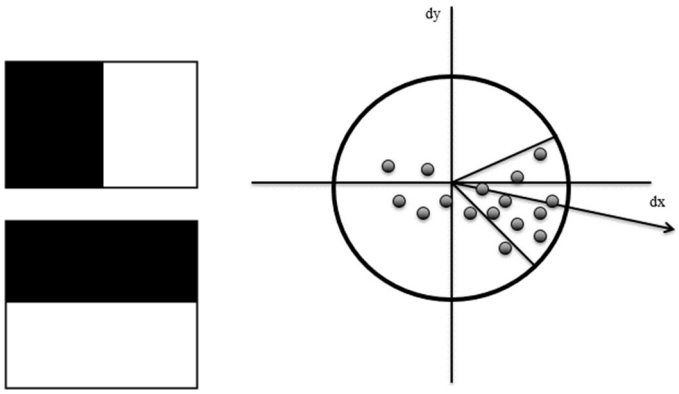


Figure 14: Haar Wavelet x and y (Left) and Orientation Assignment Calculation (Right)

A descriptor must be extracted for each interest point detected to be used in comparison to other points for the matching step. Figure 15 below shows the overall descriptor extraction process. Each circular detected point must be represented as a square region centered around the interest point. The square is then split up into smaller 4 x 4 sub regions to preserve spatial information. The Haar wavelet response is then calculated at each 5x5 spaced sample points. The responses in the x-direction (dx) and y-direction (dy) are Gaussian weighted and summed up over each sub-region to form the first components of a feature vector for an interest point detected. The next two responses will be included in a 4D descriptor vector, which is used to describe the intensity structure of the interest point. The descriptor includes the dx and dy responses along with absolute value of each. Concatenating these results for all 4x4 sub regions, gives a descriptor vector of length 64 for each interest point.

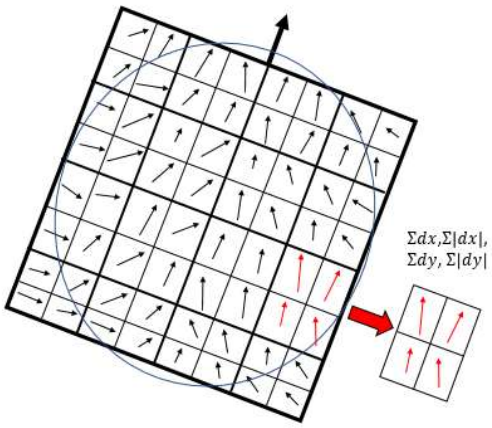


Figure 15: Feature Extraction Process to Build Descriptor

The final step is to match feature points using the extracted descriptor information. The sign of the Laplacian is used to distinguish bright intensities on dark backgrounds from dark intensities on bright backgrounds. This calculation allows for a comparison of only features with similar contrast to allow for faster matching by indexing each descriptor based on contrast. For example, in Figure 16 below, the contrast between two star objects are different so these are not considered a good match. Only features of similar contrast will be compared to allow for increase in accuracy at no computation cost. Figure 17 below will show two images of a set of SURF matched features from two separate image frames and one image matched with a rotated version of the same image. The image rotation does not need to be estimated for this application as all data sets are rotation invariant. These results are shown to demonstrates that the SURF features can be matched between two image discontinued image frames, and will be the best to use for a homography transform estimation.

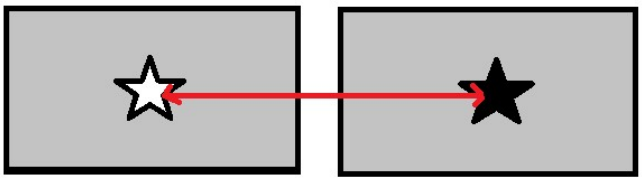


Figure 16: Bright-on-dark (left) and Dark-on-bright (right) Stars would not be Matched

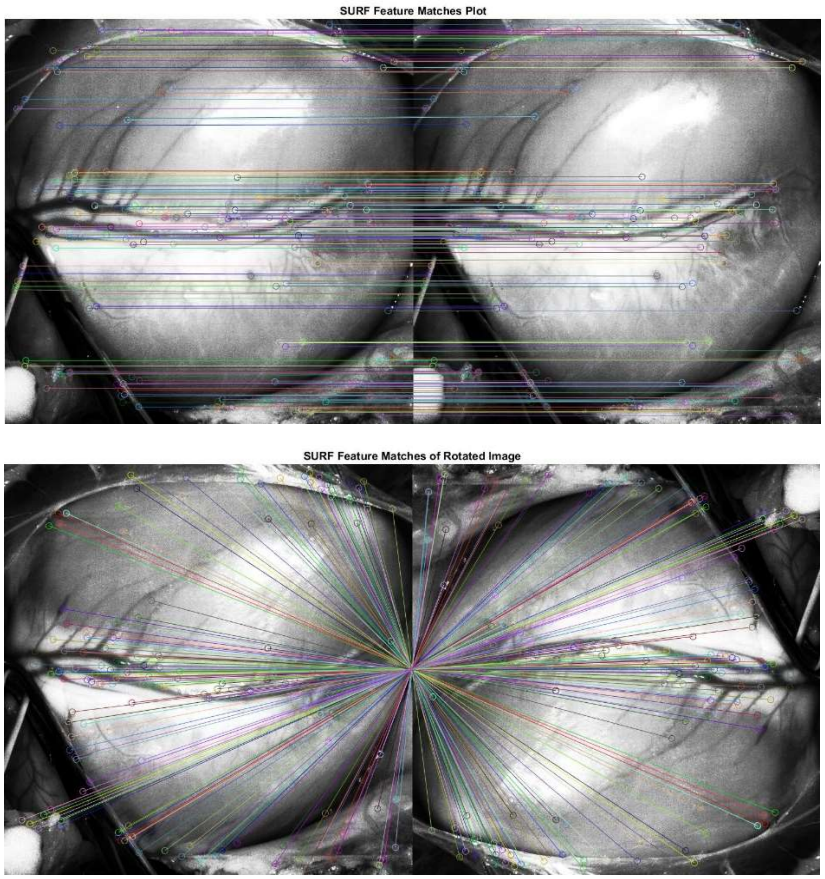


Figure 17: SURF Matched Features for Two Upright Frames (top) and Rotated Frames (bottom)

The next component to the reinitialization step is to utilize the Random Sample Consensus (RANSAC) method as an outlier rejection and homography estimation technique. RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. The mathematical model for estimating a homography transform through RANSAC can be summarized in the outline below:

1. Randomly choose N pairs
2. Find H (transfer matrix) from N pairs
    a. Where, $\overline{X_2} = H\overline{X_1}$
3. Use estimated $\widehat{H}$, where $\overline{\widehat{X_2}} = \widehat{H}\overline{X_1}$
    a. If perfect, then $\overline{\widehat{X_2}} \approx \overline{X_2}$
    b. Else, $\overline{\widehat{X_2}} - \overline{X_2} = outlier\ points$
4. If number of inliers is too small, initial H estimator is bad
5. Repeat until a large number of inliers is found.

RANSAC will be applied to SURF matched features to estimate the location of the beginning and end of the piecewise grid in one image frame to a subsequent or different transformed image. RANSAC can take a set of matched SURF points between two images, and compute a transfer matrix to relate any point within one image to the next image. The subsequent region from image one may be transformed through a homography transformation. A homography transform is defined as a non-singular 3x3 mapping matrix (H), such that for any point in an image represented by a point x that h(x)=Hx. This transformation matrix can be used to estimate motions in a two-dimensional case that are typically due to affine transforms such as translation, rotation, scaling, and others. A two-dimensional Homography matrix is calculated by the RANSAC algorithm to relate the image points (x) from frame 1 to frame 2 (x'), and can be shown in Figure 18 below. RANSAC initially takes a random set of four matched pairs, in this case SURF matches, and computes a transfer matrix between them. The initial [3x3] matrix is then applied to all the matched points from image one to compute an estimate of the location of the input data in image two. If there is a difference between the estimate and actual matched pairs, then the difference is labeled as outliers and must be minimized. RANSAC will continue to iterate until the maximum amount of inlier data points is found. A data point will be considered an inlier if its geometric distance to the estimate, also known as error, is minimized (Choi et. al, 2009). If there are fewer correct SURF matched features, more iterations are required to increase the probability that a selected subset will contain only correct matches (Hassner et. al, 2013). It is important to make sure the SURF matches found are correct to reduce the number of RANSAC iterations necessary. The RANSAC homography transform results can be applied to any set of points within one image frame, and project these points to their locations in the next image frame. To reinitialize the tracking algorithm due to a failed estimate, the SURF features will be combined with RANSAC to estimate a homography transform that will move the beginning and end of the piecewise grid. With a new start and stop point, the piecewise grid will be redefined by updating the corners and edges found in each patch. The piecewise motion estimation will then return to the KLT tracker to estimate the motion in the next image frame.

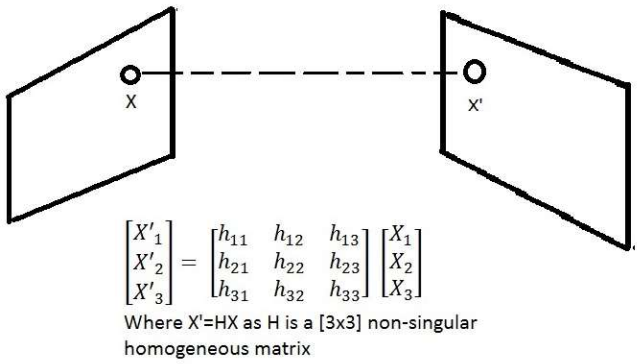


Figure 18: Example Homography Transform Between Two Planes x and x'

3.3: Experimental Design

The piecewise tracking approach explained above was tested on five heart motion data sets following the steps presented in this section. First, both piecewise motion constraints were validated over a small portion of data, and this function, labeled as "DeterminePatchNum4.m," is attached in the Appendix. This function looped through different patch numbers and tested whether the motion from frames one to two is normally distributed within each patch. In addition, this function looped through a set number of image frames to check if the motion between patch neighbors is normally distributed, and record the differences in the horizontal and vertical directions. After the validation step, thresholds were then calculated using the standard deviation values.

Next, the piecewise tracking algorithm was tested across all image frames of each data set, to prove that this approach will track a ROI continuously. With the patch number and motion thresholds determined for each data set during initialization, the corners and edges of the ROI were grouped into each patch. The KLT tracker was then initialized, and the edge locations were moved frame by frame using the estimated motion. The motion of any patch in the U and V direction was also flagged as a failed tracking estimate if either constraint was not met. To improve the tracking results, any flagged patch, due to the flow difference with its neighbor being higher than the set threshold, was set to follow the motion of its neighbor for that frame. This step was done to ensure the ROI to be tracked was not lost, and to keep the edges from separating completely. If the number of flagged image frames reached three, the reinitialization step was completed. To begin, the previous image frame with the most matching SURF features

with the current discontinued frame was determined using the “DetermineBestFit2.m” function, which can also be found in the Appendix. The two image frames were then used to calculate a homography transform through the RANSAC algorithm with SURF matched features between them. The homography was then used to move the current start and stop points to realign the piecewise grid. The corner features and edges were then resampled into new patch locations, and the KLT tracker was restarted.

CHAPTER 4: RESULTS

4.1: Initialization

The results of the optical flow measurements in the U (x) and V (y) directions within a patch was checked by the chi-square goodness of fit test, and these results are shown below in Tables 1-5. The tables below show the total number of patches with a motion distribution that failed the chi-square test and are not normally distributed. When both the U and V motion was found to be normally distributed within every patch, the patch number was set. The table also includes the four motion thresholds calculated as the maximum offset allowed for motion among neighboring patches in either the U or V directions. These results verify that the first constraint for the piecewise tracking is met. It also provides results to support the second constraint that the motion between neighbors is continuous if the difference between them is lower than the threshold limit.

For data sets 1, 2, 3, and 5 the patch number was determined to be 36, while patch 4 was determined to be 25.

Table 1: Initialization of the Patch Number and Thresholds for Data Set 1

Data Set 1 - Within Patch Test		
Patch Number	# Fails U Flow Chi^2 Test	# Fails V Flow Chi^2 Test
16	5	4
25	2	2
36	0	0
Neighbor Motion Test	Thresholds	
U Horiz	3.9795	
U Vert	5.077	
V Horiz	2.7776	
V Vert	3.2662	

Table 2: Initialization of the Patch Number and Thresholds for Data Set 2

Data Set 2 - Within Patch Test		
Patch Number	# Fails U Flow Chi^2 Test	# Fails V Flow Chi^2 Test
16	5	3
25	2	0
36	0	0
Neighbor Motion Test	Thresholds	
U Horiz	2.0307	
U Vert	1.8698	
V Horiz	3.1173	
V Vert	2.3241	

Table 3: Initialization of the Patch Number and Thresholds for Data Set 3

Data Set 3 - Within Patch Test		
Patch Number	# Fails U Flow Chi^2 Test	# Fails V Flow Chi^2 Test
16	1	3
25	2	2
36	0	0
Neighbor Motion Test	Thresholds	
U Horiz	13.0212	
U Vert	10.7248	
V Horiz	7.3963	
V Vert	7.1009	

Table 4: Initialization of the Patch Number and Thresholds for Data Set 4

Data Set 4 - Within Patch Test		
Patch Number	# Fails U Flow Chi^2 Test	# Fails V Flow Chi^2 Test
16	5	4
25	0	0
Neighbor Motion Test	Thresholds	
U Horiz	0.7936	
U Vert	0.7564	
V Horiz	1.0184	
V Vert	0.5009	

Table 5: Initialization of the Patch Number and Thresholds for Data Set 5

Data Set 5 - Within Patch Test		
Patch Number	# Fails U Flow Chi^2 Test	# Fails V Flow Chi^2 Test
16	2	4
25	1	2
36	0	0
Neighbor Motion Test	Thresholds	
U Horiz	9.3772	
U Vert	3.6902	
V Horiz	2.9781	
V Vert	1.7882	

The distributions of all the neighboring differences are shown in Figures 19-38 as histograms for each of the five data sets. The distributions shown appear to be close to normal distribution, as expected. The two-sigma values were calculated as a motion threshold for the horizontal and vertical neighbors in the U and V directions. If any difference between neighboring patches were to be greater than its respective threshold, the frame will be flagged as a failed motion estimate.

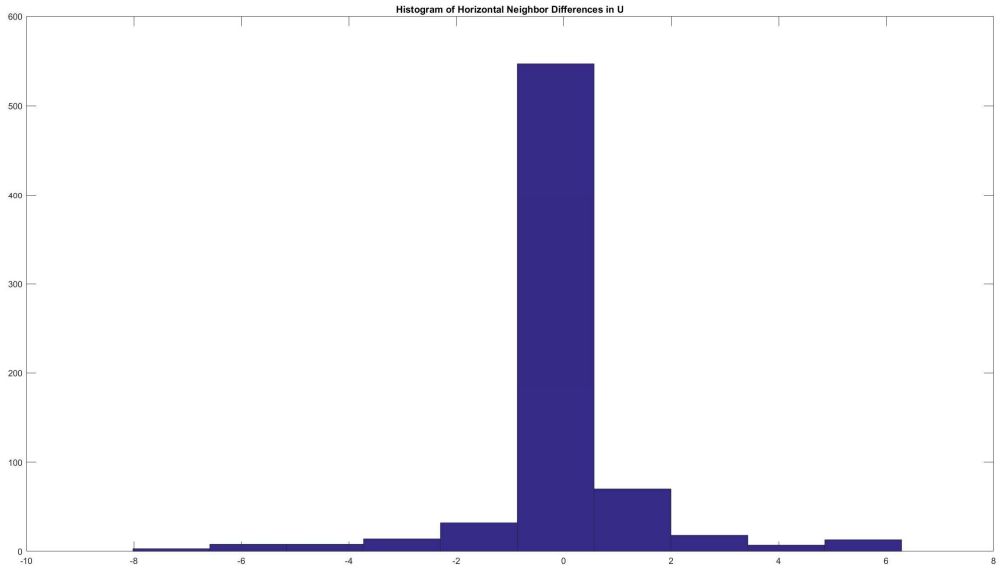


Figure 19: Data Set 1 Histogram of U Patch Neighbors Horizontal Direction

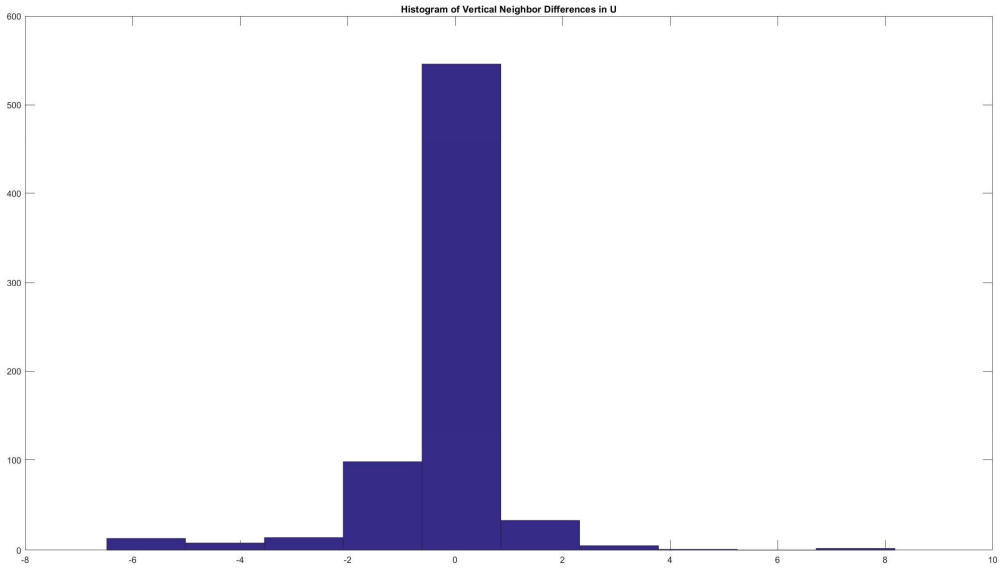


Figure 20: Data Set 1 Histogram of U Patch Neighbors Vertical Direction

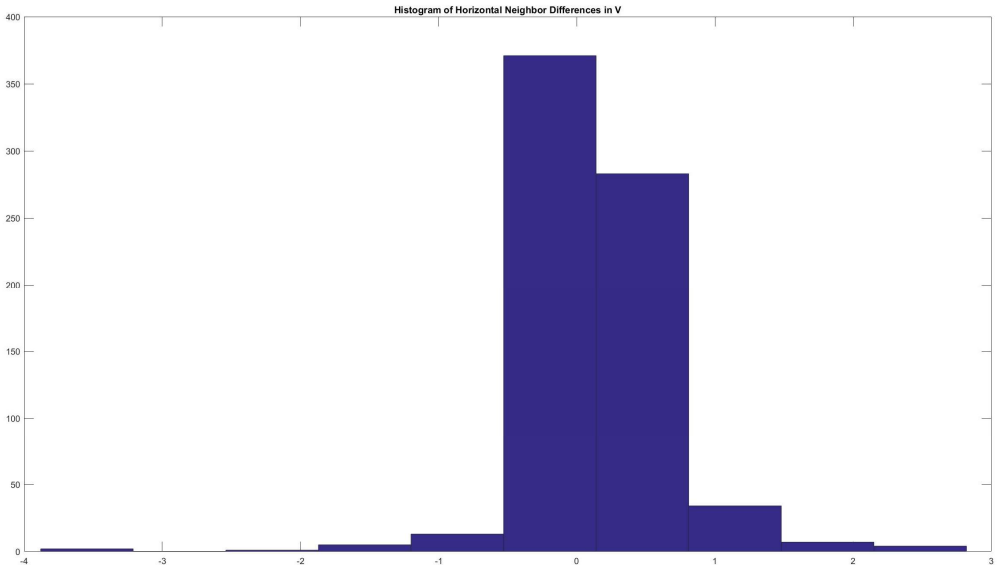


Figure 21: Data Set 1 Histogram of V Patch Neighbors Horizontal Direction

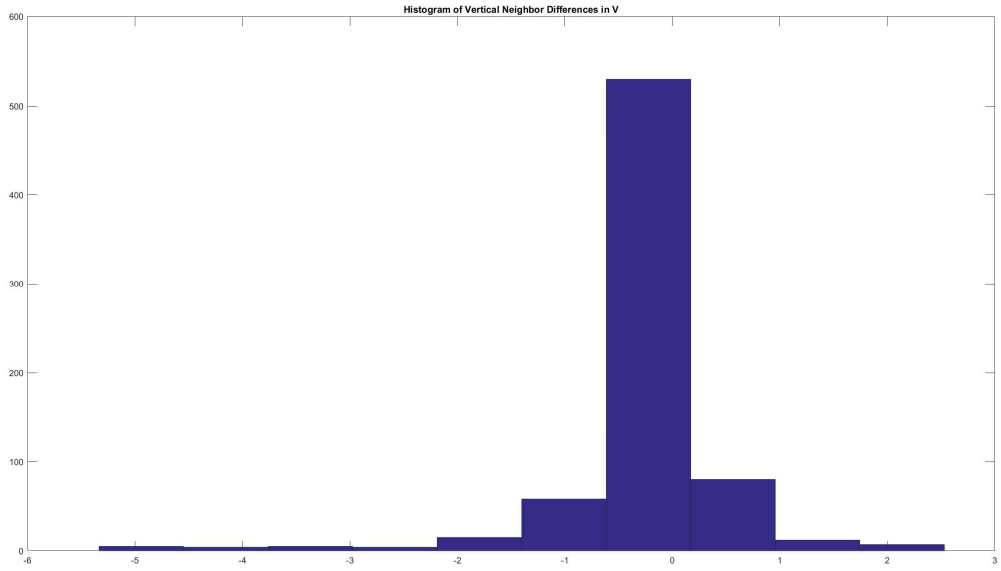


Figure 22: Data Set 1 Histogram of V Patch Neighbors Vertical Direction

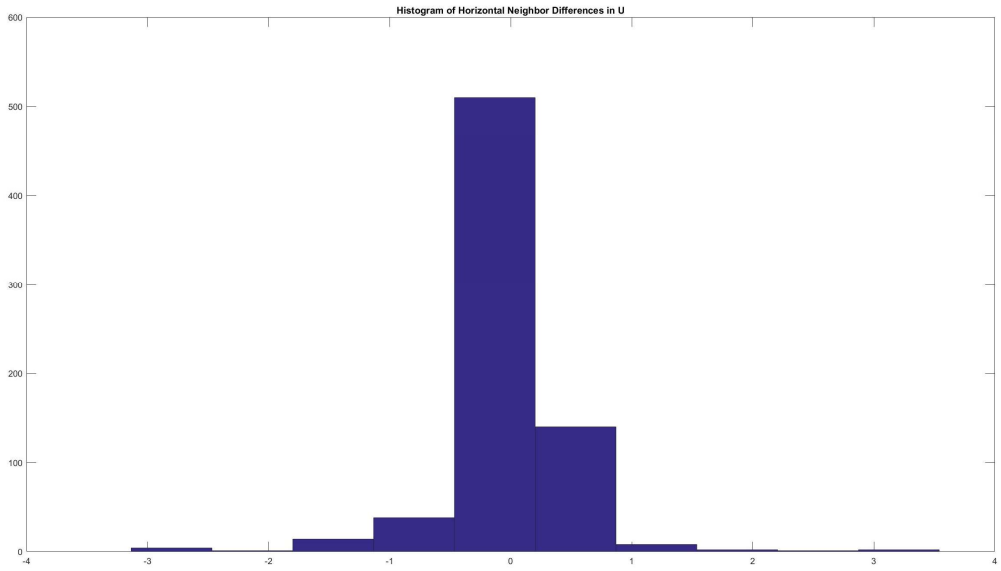


Figure 23: Data Set 2 Histogram of U Patch Neighbors Horizontal Direction

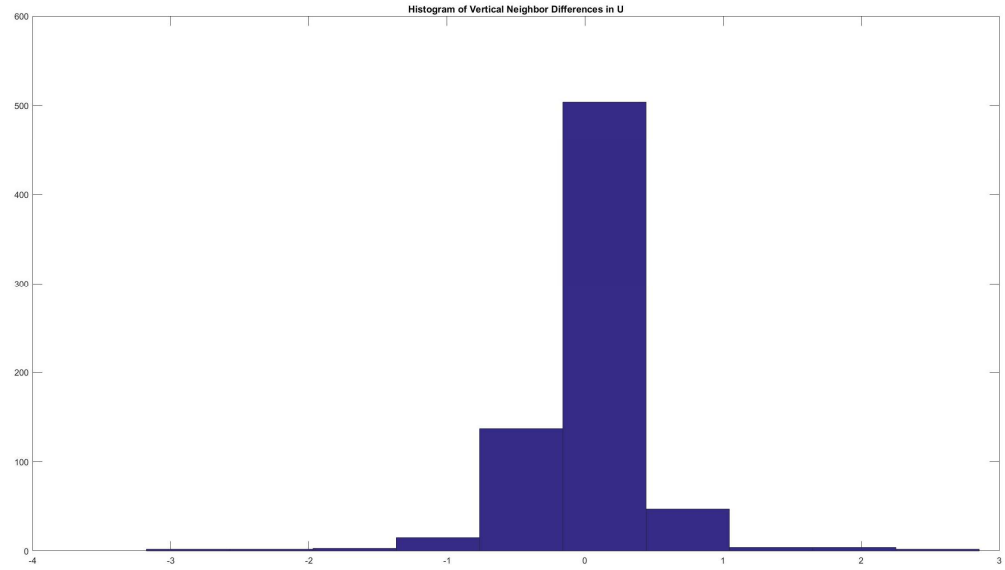


Figure 24: Data Set 2 Histogram of U Patch Neighbors Vertical Direction

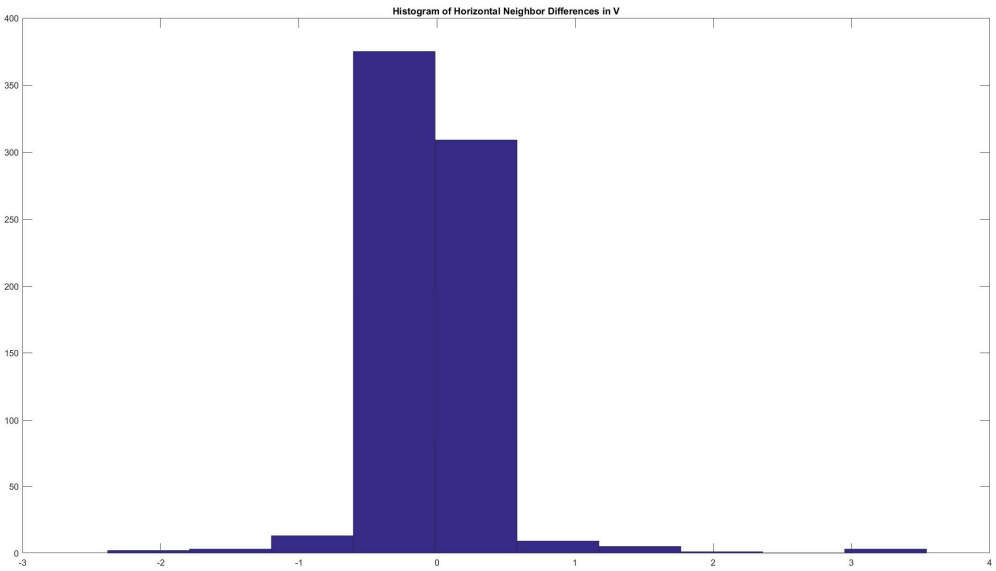


Figure 25: Data Set 2 Histogram of V Patch Neighbors Horizontal Direction

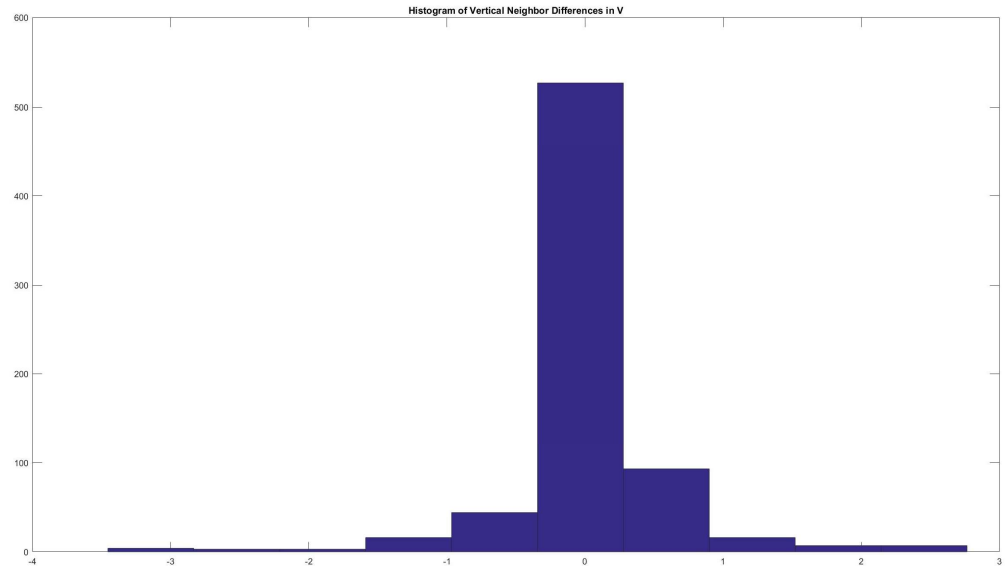


Figure 26: Data Set 2 Histogram of V Patch Neighbors Vertical Direction

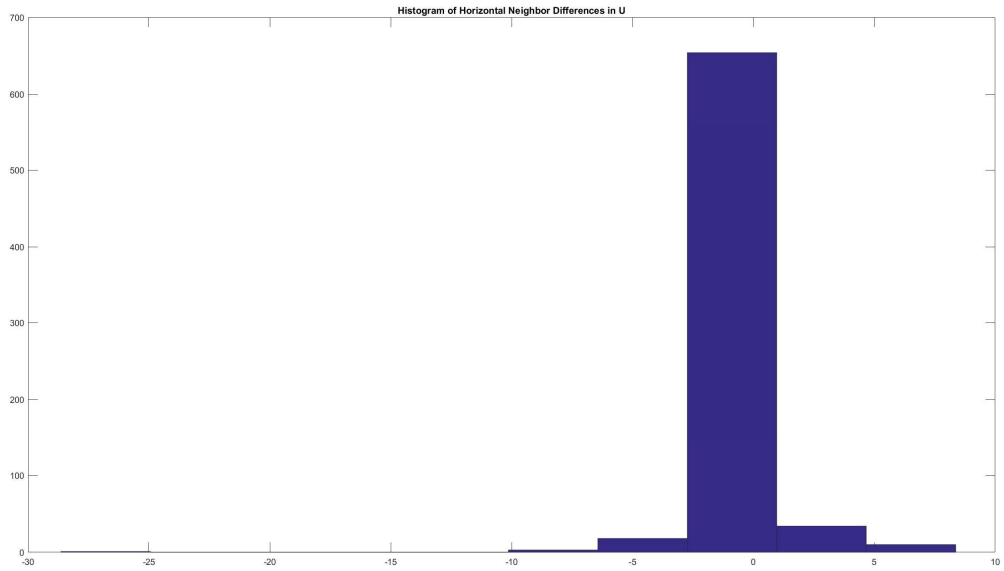


Figure 27: Data Set 3 Histogram of U Patch Neighbors Horizontal Direction

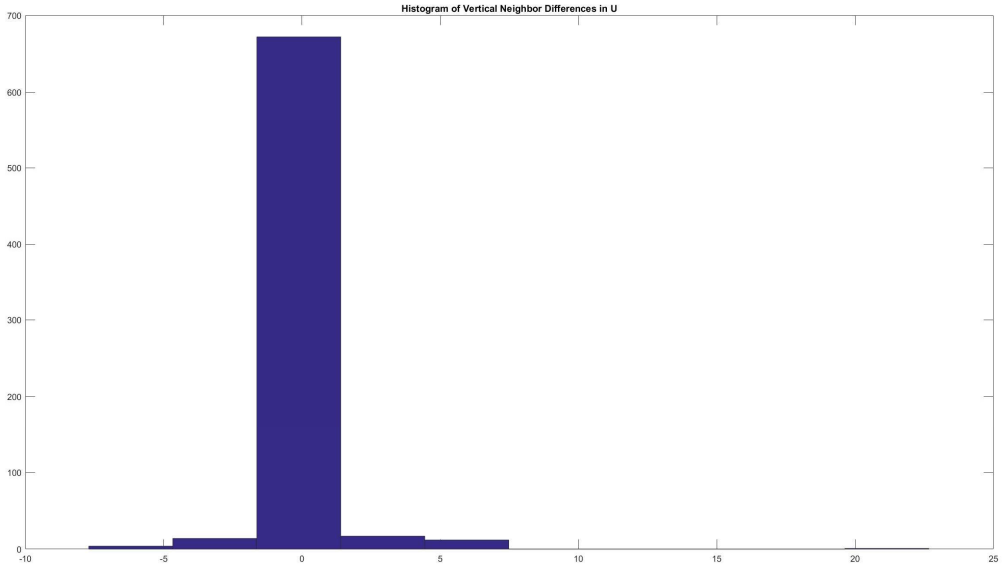


Figure 28: Data Set 3 Histogram of U Patch Neighbors Vertical Direction

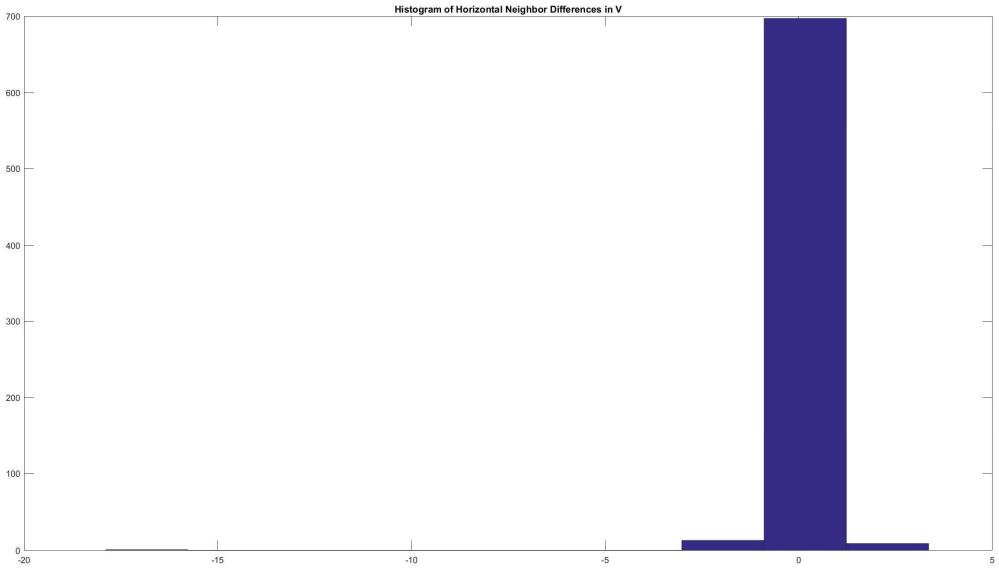


Figure 29: Data Set 3 Histogram of V Patch Neighbors Horizontal Direction

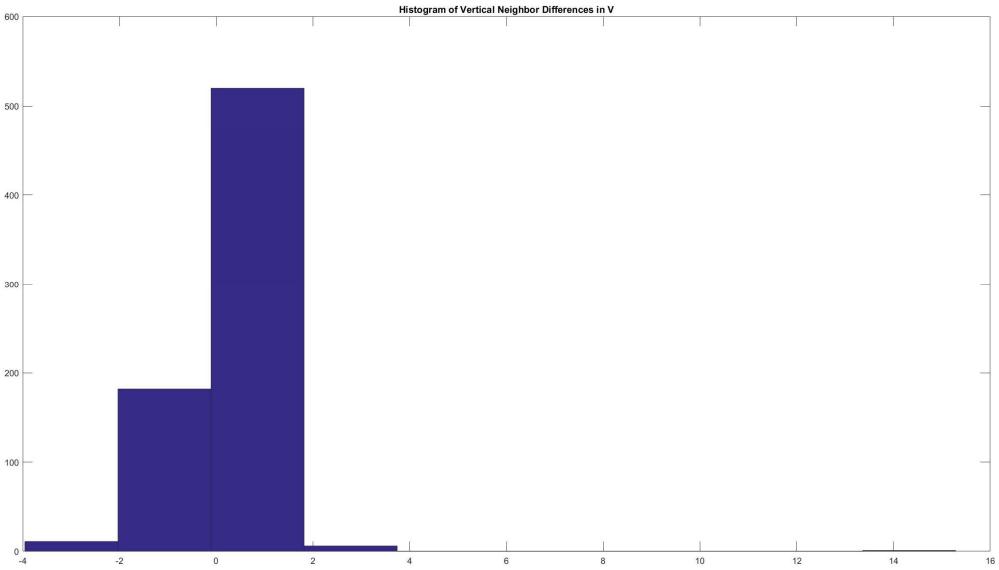


Figure 30: Data Set 3 Histogram of V Patch Neighbors Vertical Direction

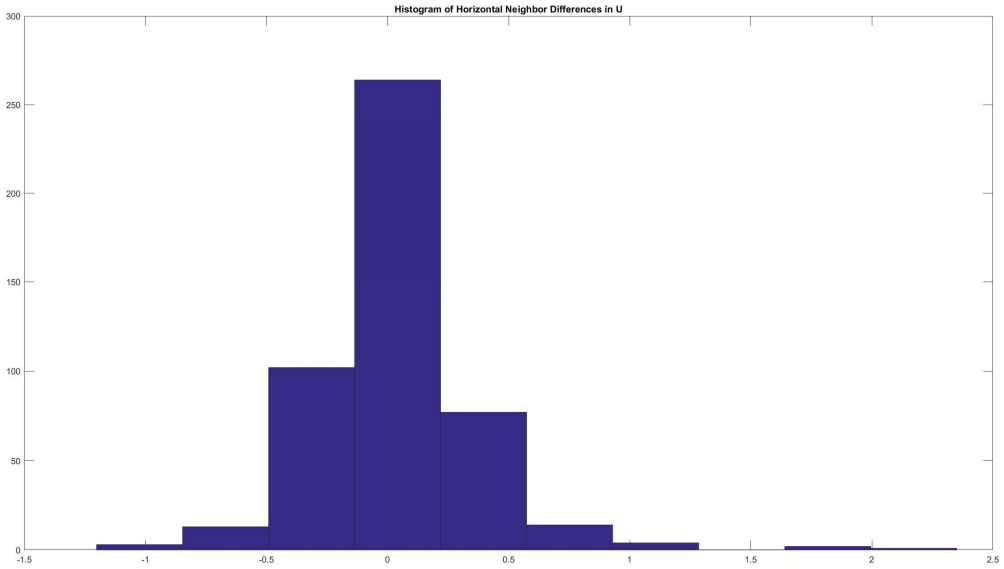


Figure 31: Data Set 4 Histogram of U Patch Neighbors Horizontal Direction

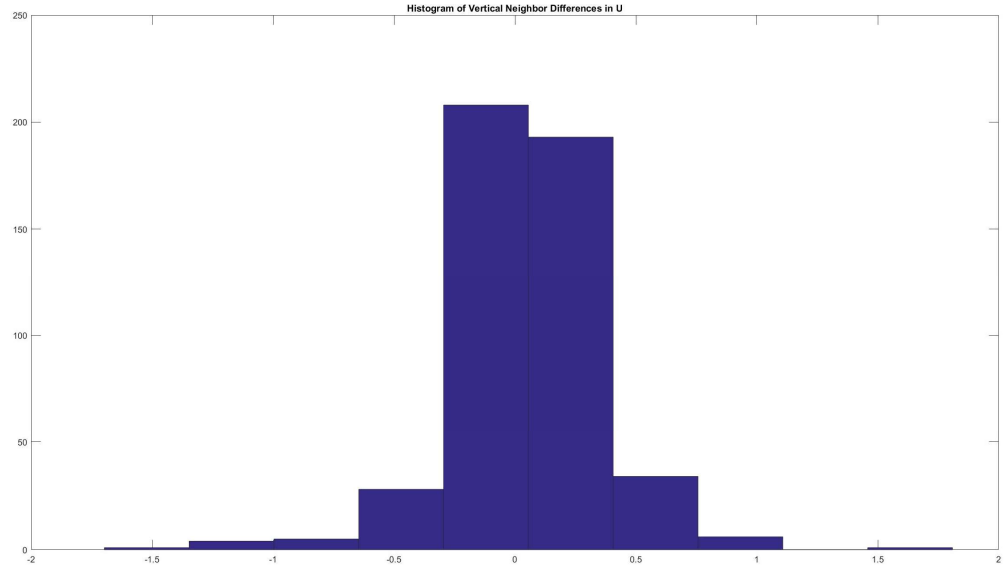


Figure 32: Data Set 4 Histogram of U Patch Neighbors Vertical Direction

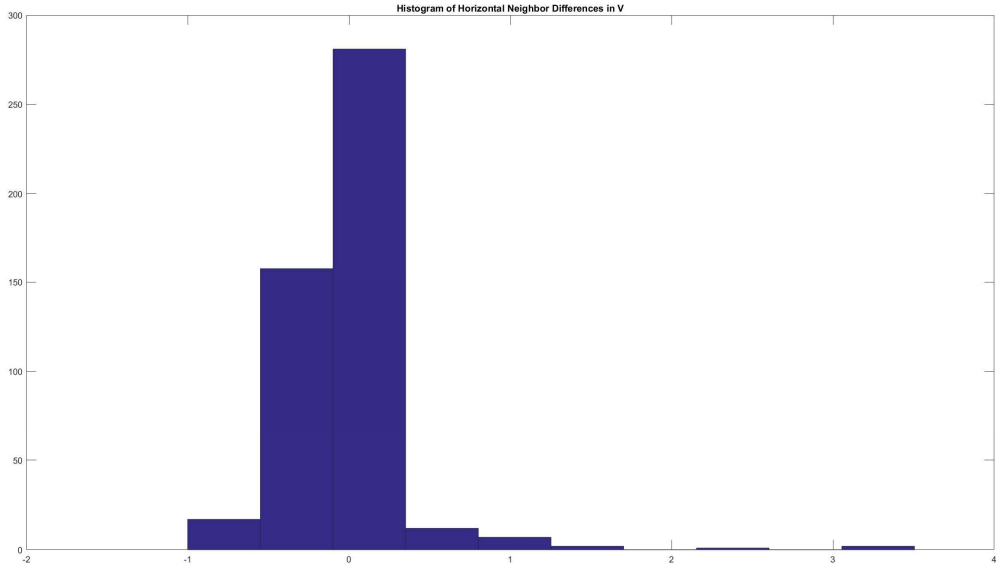


Figure 33: Data Set 4 Histogram of V Patch Neighbors Horizontal Direction

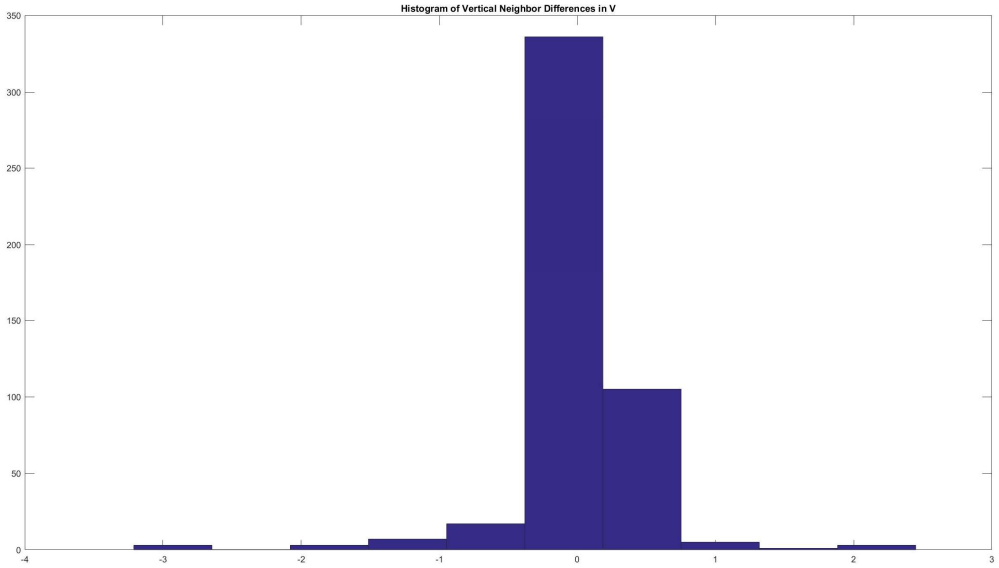


Figure 34: Data Set 4 Histogram of V Patch Neighbors Vertical Direction

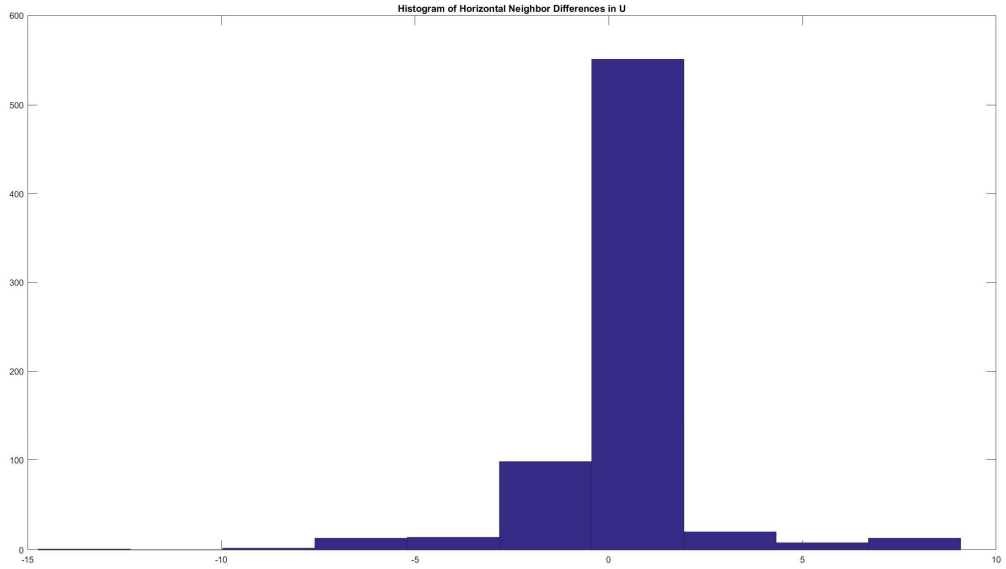


Figure 35: Data Set 5 Histogram of U Patch Neighbors Horizontal Direction

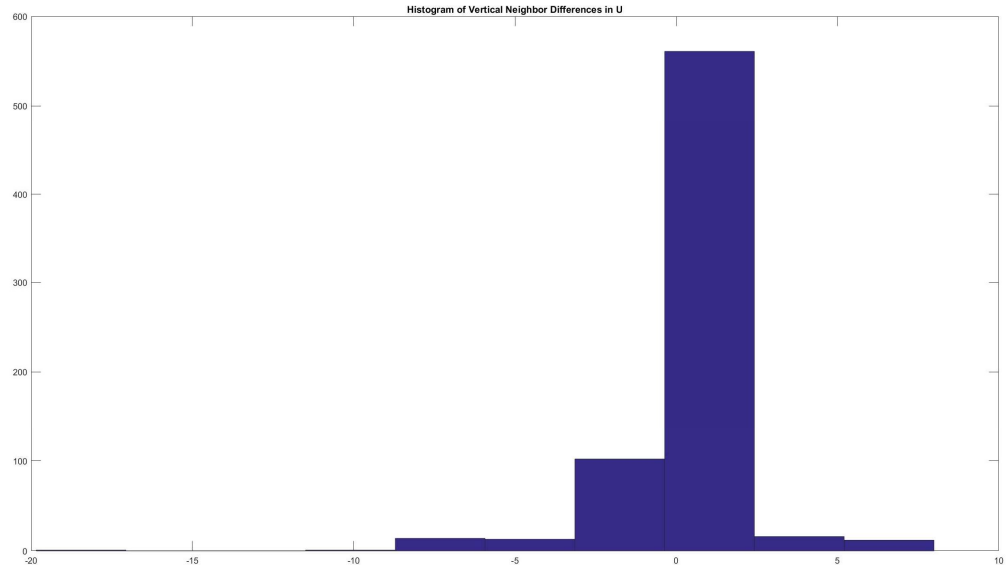


Figure 36: Data Set 5 Histogram of U Patch Neighbors Vertical Direction

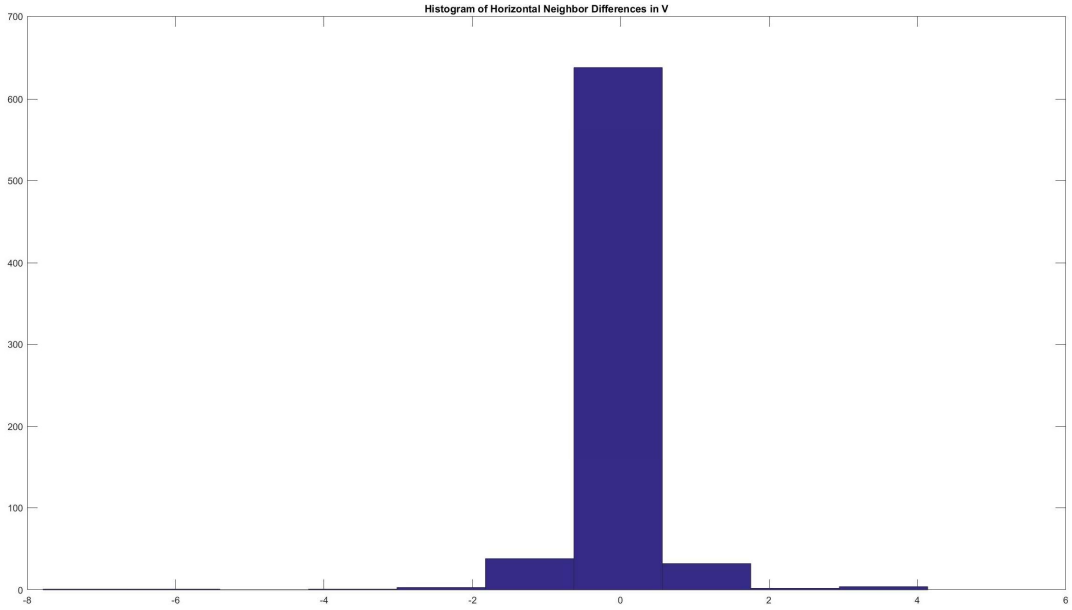


Figure 37: Data Set 5 Histogram of V Patch Neighbors Horizontal Direction

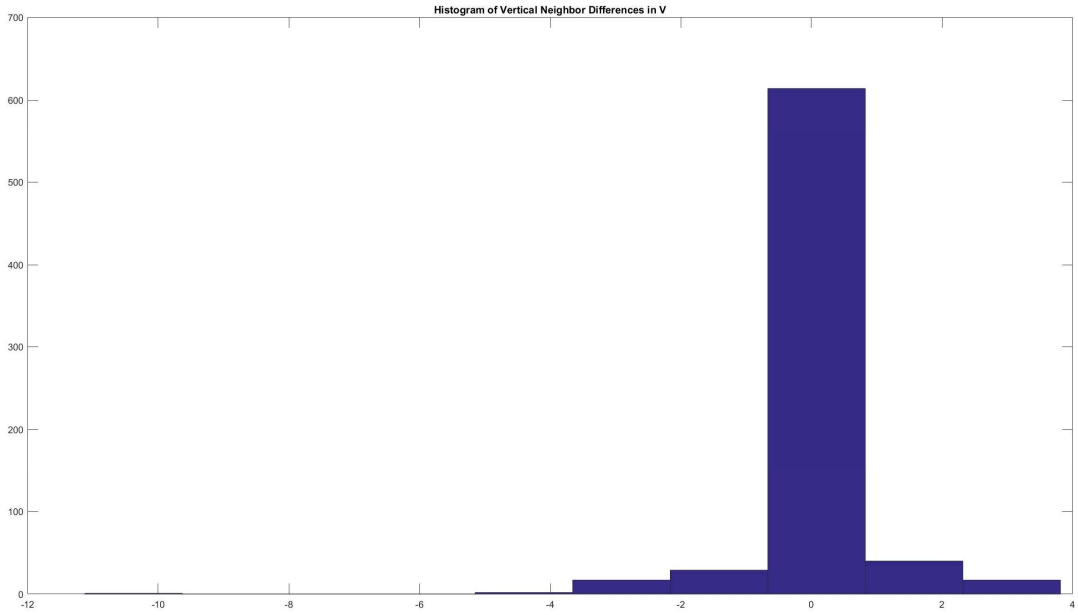


Figure 38: Data Set 5 Histogram of V Patch Neighbors Vertical Direction

4.2: Tracking and Reinitialization

The tracking results of data set 1 can be shown in Figure 39 below, including the initial frame (frame 1) that contains the initializing patch grid, with frames 2, 1537-1540. This whole set contains 2430 image frames in total, and frames 1537-1540 are chosen to demonstrate the tracking results in the middle of the dataset set. Throughout the entire data set, the total number of resets was found to be 409 times or 16.8% of the total number of frames. An example of this reinitialization is also provided in the figure from frame 1539 to 1540. The average optical flow measure for each patch is also shown in Figures 40 and 41 below along with the corrected measure of every patch that does not pass the set thresholds. These results show that there exists a distinct pattern to the movement of the ROI. The flow movement was found to increase and then always returns to its original position with a zero-flow measure at the end of every heart cycle. A flagged frame is also highlighted by a black star to show when the optical flow of one or more patches is considered an outlier that does not pass either piecewise constraint. It is important to note that this is the only data set where a complete occlusion of the ROI occurs around frames 1700 and 2000. The tracking algorithm was not able to correct itself with a complete blockage of the region. These results are shown in Figure 42 below with the complete block pushing the tracking estimate away from the actual location. After the complete blockage of the region, the actual image changes as the region cannot be distinguished from the blockage. This change in the image causes the reinitialization step to not be able to realign and restart the tracking process. With these frames taken out of the data set, the algorithm is able to completely track the ROI over all other image frames.

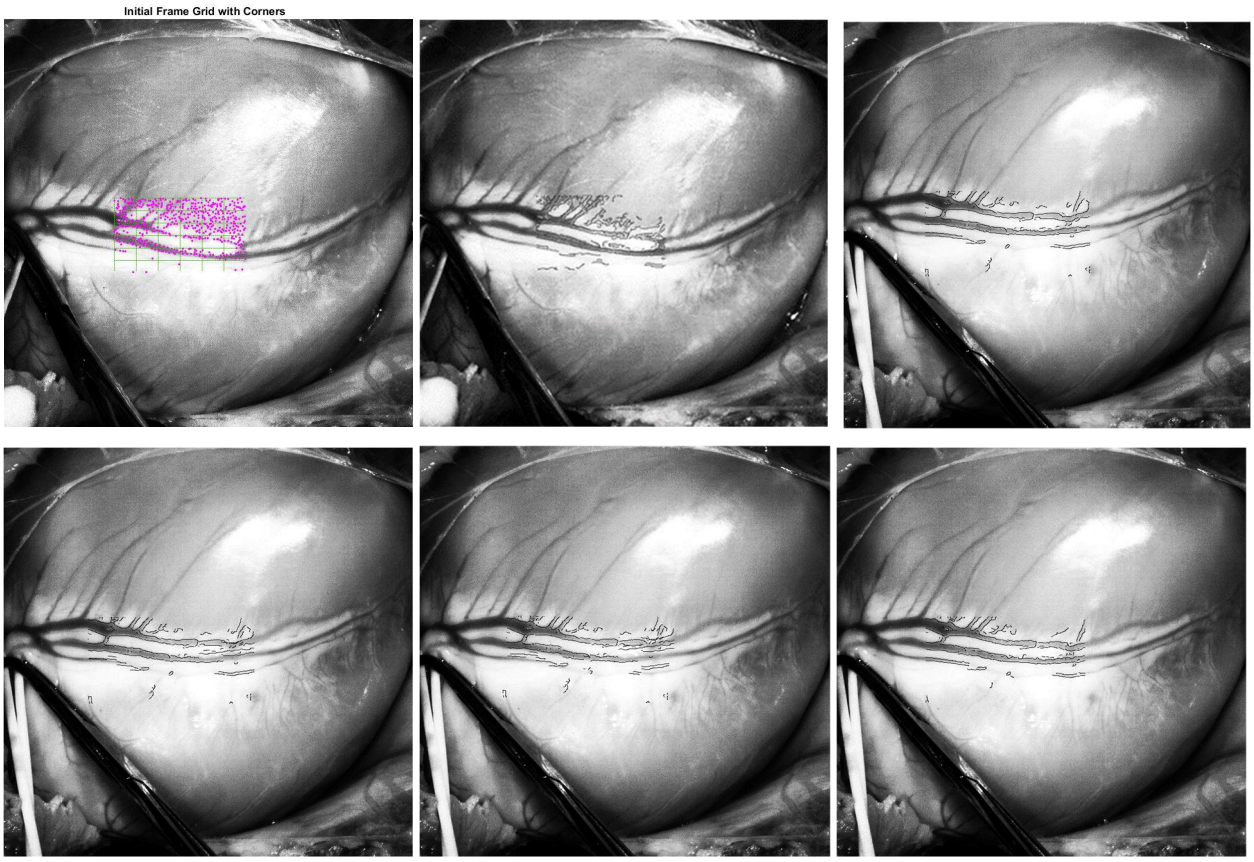


Figure 39: Data Set 1 Tracking Results Starting from Frame 1, 2, 1537 (top) to Frame 1538-1540 (bottom)

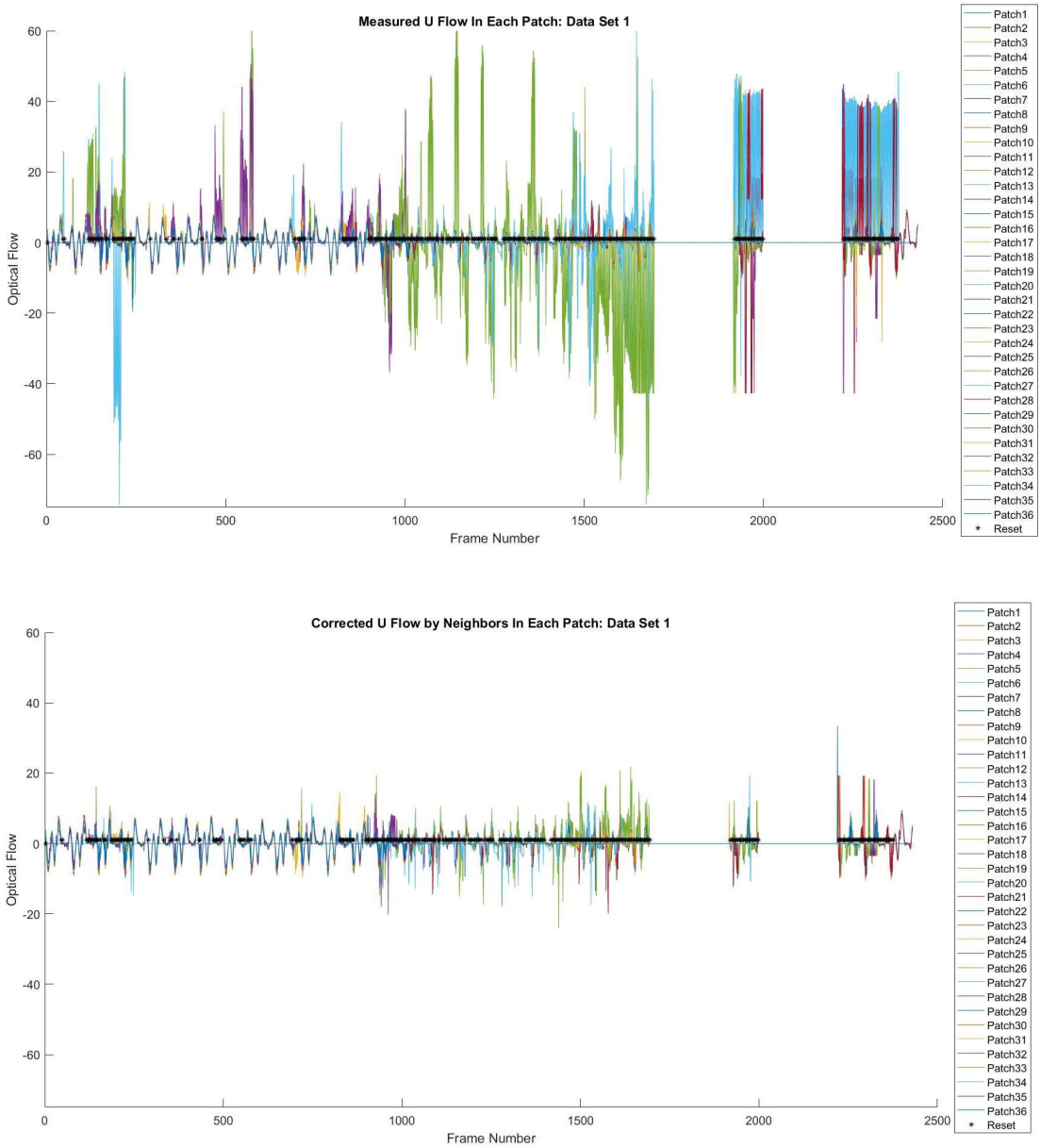


Figure 40: Measured U Optical Flow vs Corrected Flow by Neighbors

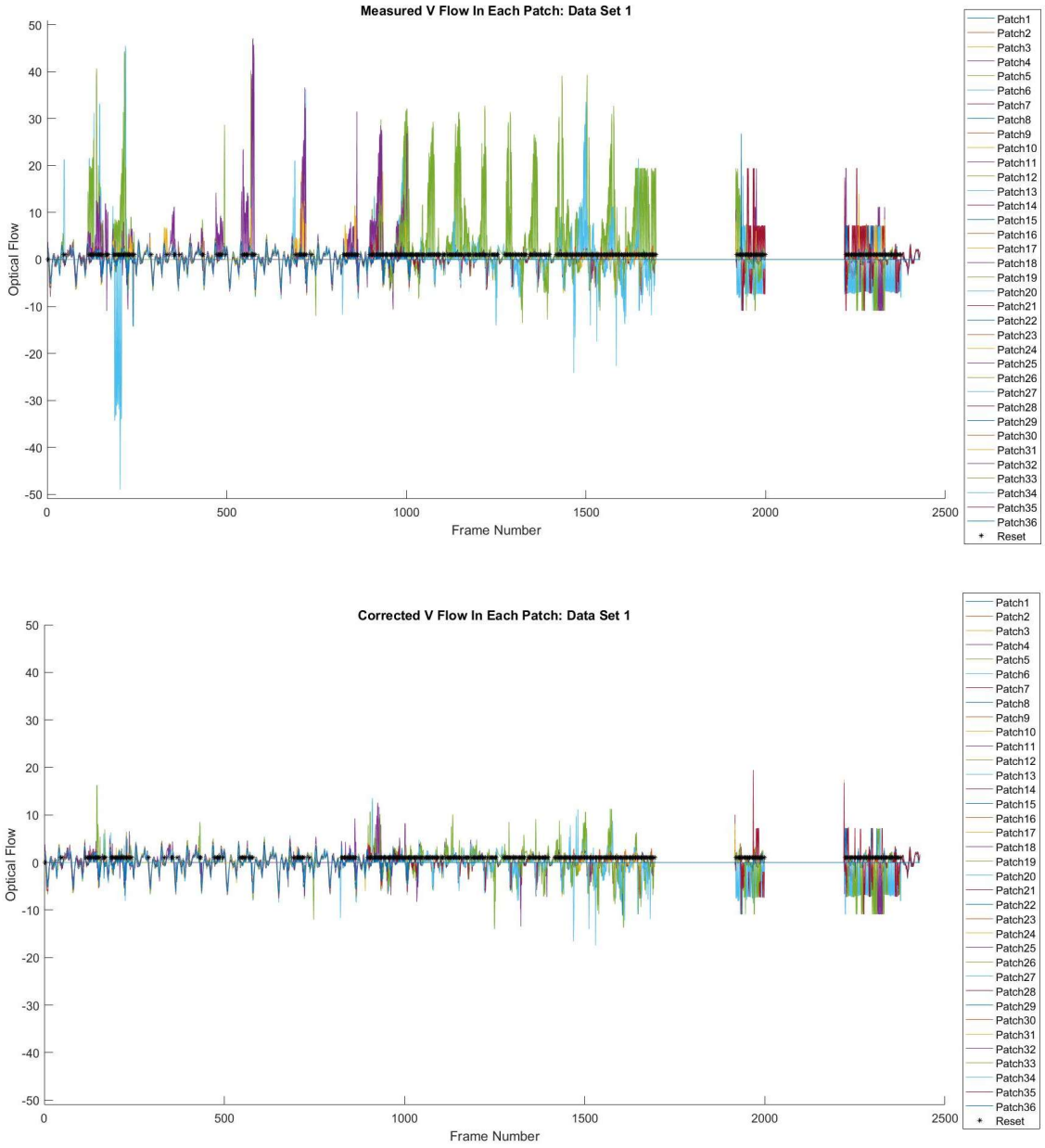


Figure 41: Measured V Optical Flow vs Corrected Flow by Neighbors

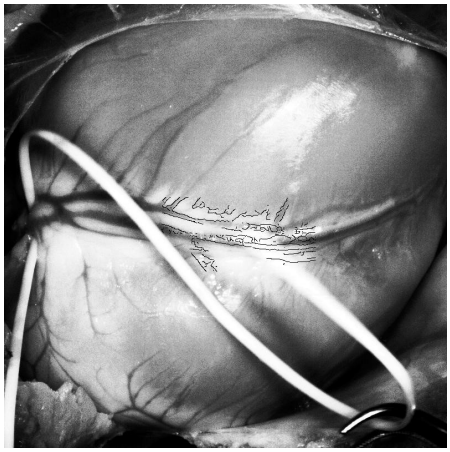


Figure 42: Frame 1768 Occlusion

For data set 2, we found that the ROI was tracked continuously in all image frames. Figure 43 below shows some results starting with frame 1, 2, and 1607 in the top row followed by 1608-1610 in the bottom row. The flagged image frames start at 1608 as the left edge of the blood vessel separates from its actual boundary location, and is then realigned due to the reinitialization step. This shows that the motion thresholds calculated are accurate to pick up any small change between two patch neighbors that would break the piecewise motion estimate. In this data set, the total number of resets was found to be 540 times or 22.2% of the total number of frames. The average optical flow measure for each patch is also shown in Figures 44 and 45 below along with the corrected measure of every patch by their respective neighboring flow. A flagged frame is also highlighted by a black star to show when the optical flow one or more patches is considered an outlier that does not pass either piecewise constraint.

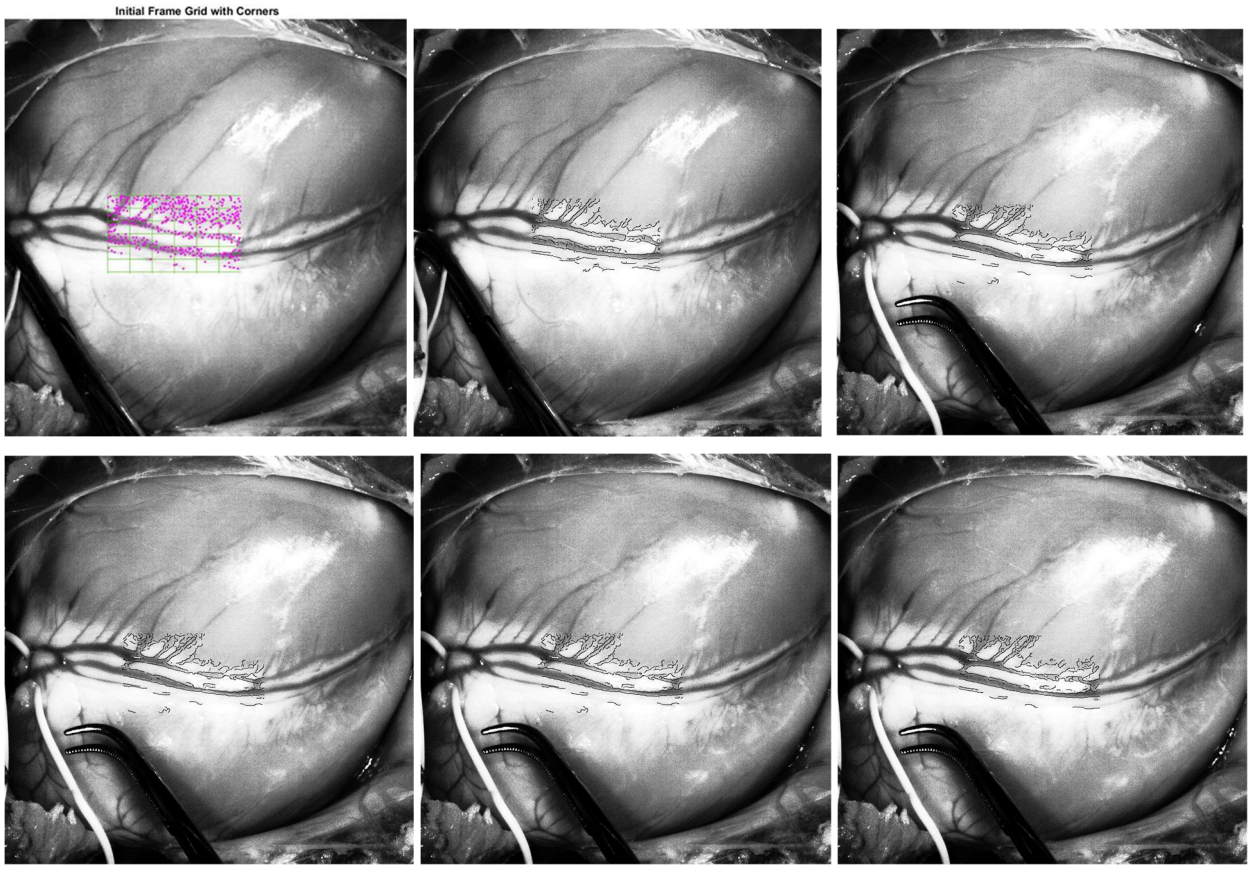


Figure 43: Data Set 2 Tracking Results Starting from Frame 1, 2, 1607 (top) to Frame 1608-1610 (bottom)

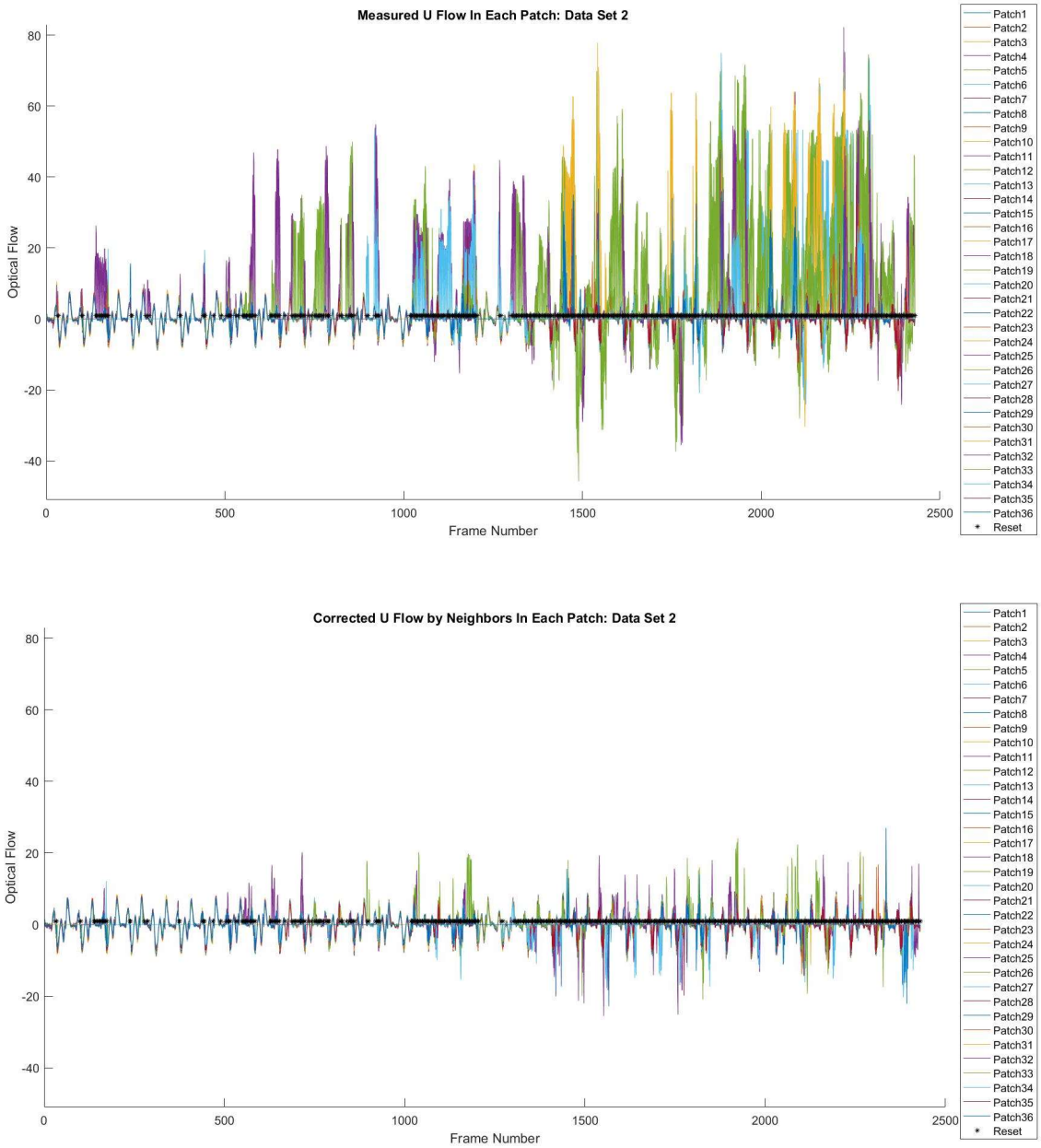


Figure 44: Measured U Optical Flow vs Corrected Flow by Neighbors

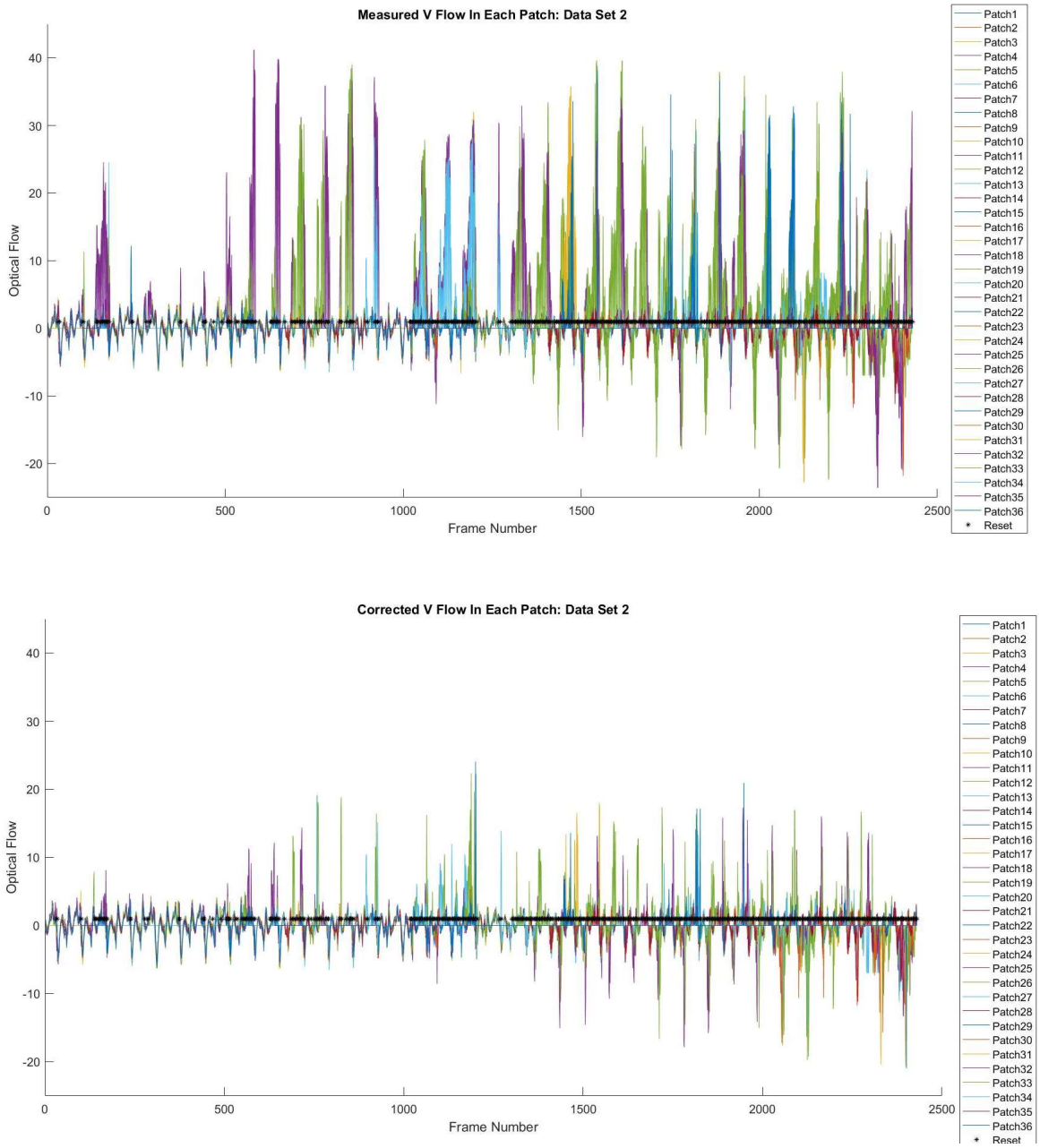


Figure 45: Measured V Optical Flow vs Corrected Flow by Neighbors

The ROI was also continuously tracked for data set 3 over a set number of 2430 frames. Figure 46 below shows the tracking and reinitialization from frames 1, 2, and 191-194. These results show that the rightmost edges of the vessel break off from their actual location and then are reset by the last frame. In this data set, the total number of resets was found to be 419 times or 17.2% of the total number of frames. The average optical flow measure for each patch is also shown in Figures 47 and 48 below along with the corrected measure of every patch by their

respective neighboring flow. A flagged frame is also highlighted by a black star to show when the optical flow one or more patches is considered an outlier that does not pass either piecewise constraint. These results also show that there exists a distinct pattern to the movement of the ROI as the flow movement increases and then always returns to its original position with a zero flow at the end of every heart cycle.

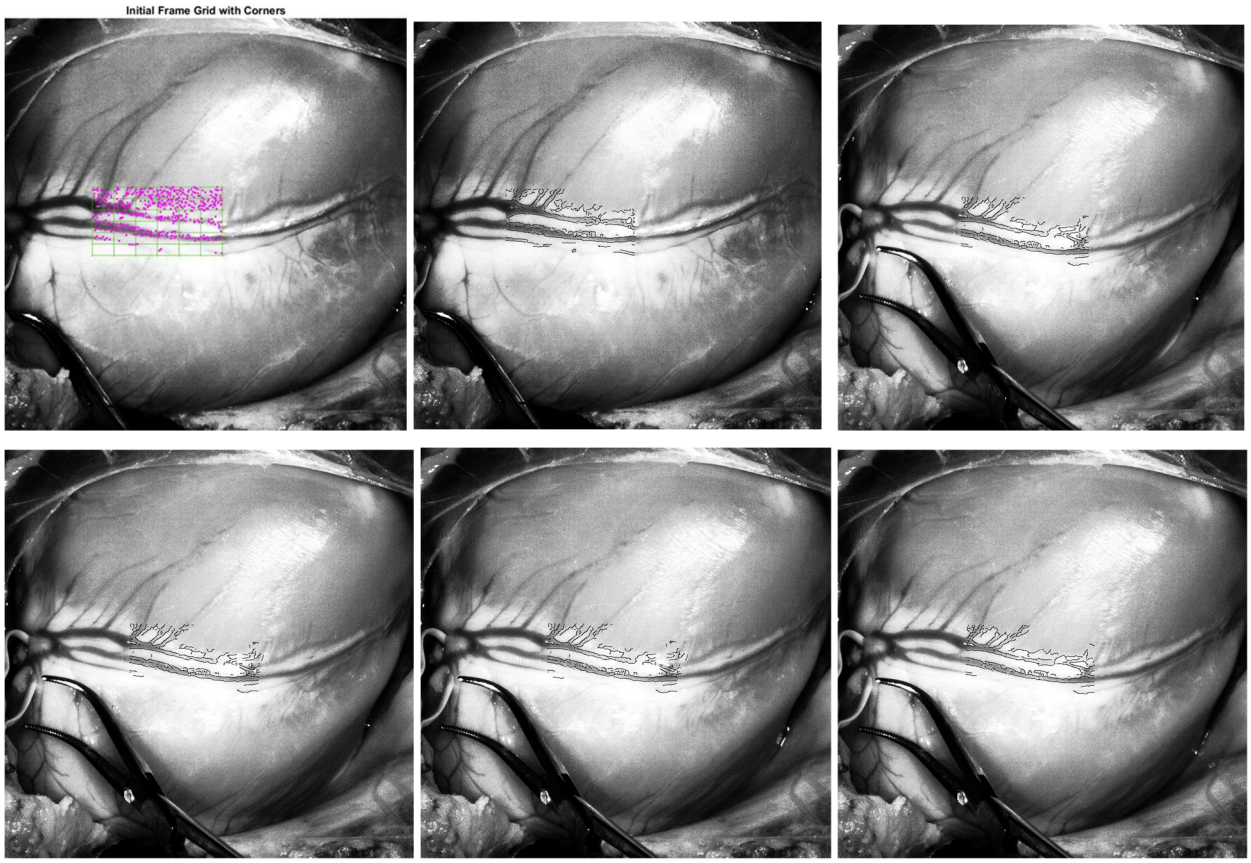


Figure 46: Data Set 3 Tracking Results Starting from Frame 1, 2, 191 (top) to Frame 192-194 (bottom)

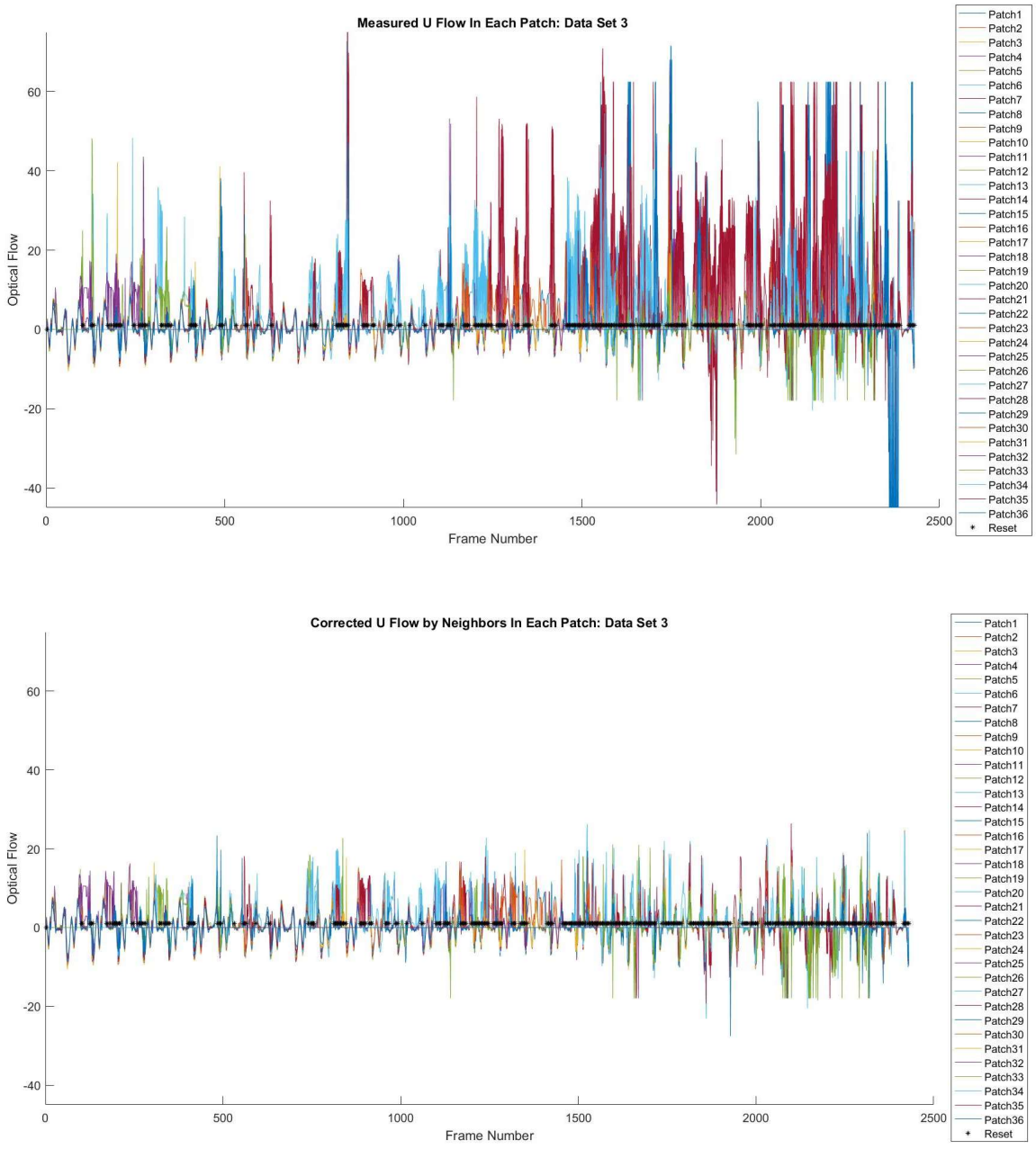


Figure 47: Measured U Optical Flow vs Corrected Flow by Neighbors

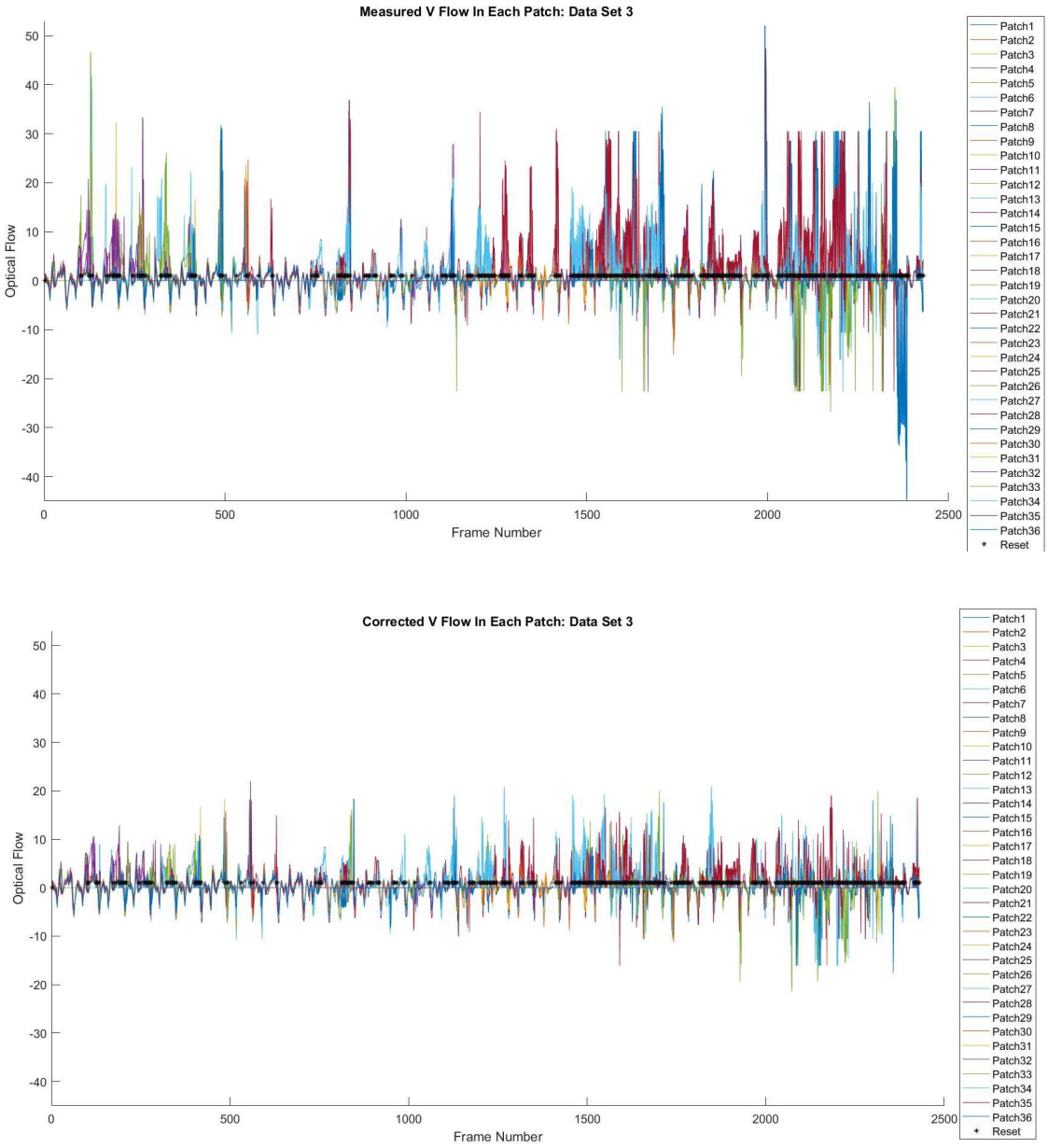


Figure 48: Measured V Optical Flow vs Corrected Flow by Neighbors

Data set 4 provides a different result as the number of patches changes to 25, but the algorithm was still able to continuously track the ROI continuously over 2400 frames. Figure 49 below shows the tracking and reinitialization from frames 1, 2, and 2354-2357. It can be shown in frames 2354-2357 the bottom left edges of the blood vessel fall off track and are eventually moved back to their correct position. This shows that the algorithm can recover from more than one offset between neighboring patches and restart the tracking process. Overall, the total

number of resets was found to be 739 times or 30.8% of the total number of frames. The average optical flow measure for each patch is also shown in Figures 50 and 51 below along with the corrected measure of every patch that does not pass the set thresholds. These results show that there exists a distinct pattern to the movement of the ROI as the flow movement increases and then always returns to its original position with a zero flow at the end of every heart cycle. A flagged frame is also highlighted by a black star to show when the optical flow one or more patches is considered an outlier that does not pass either piecewise constraint.

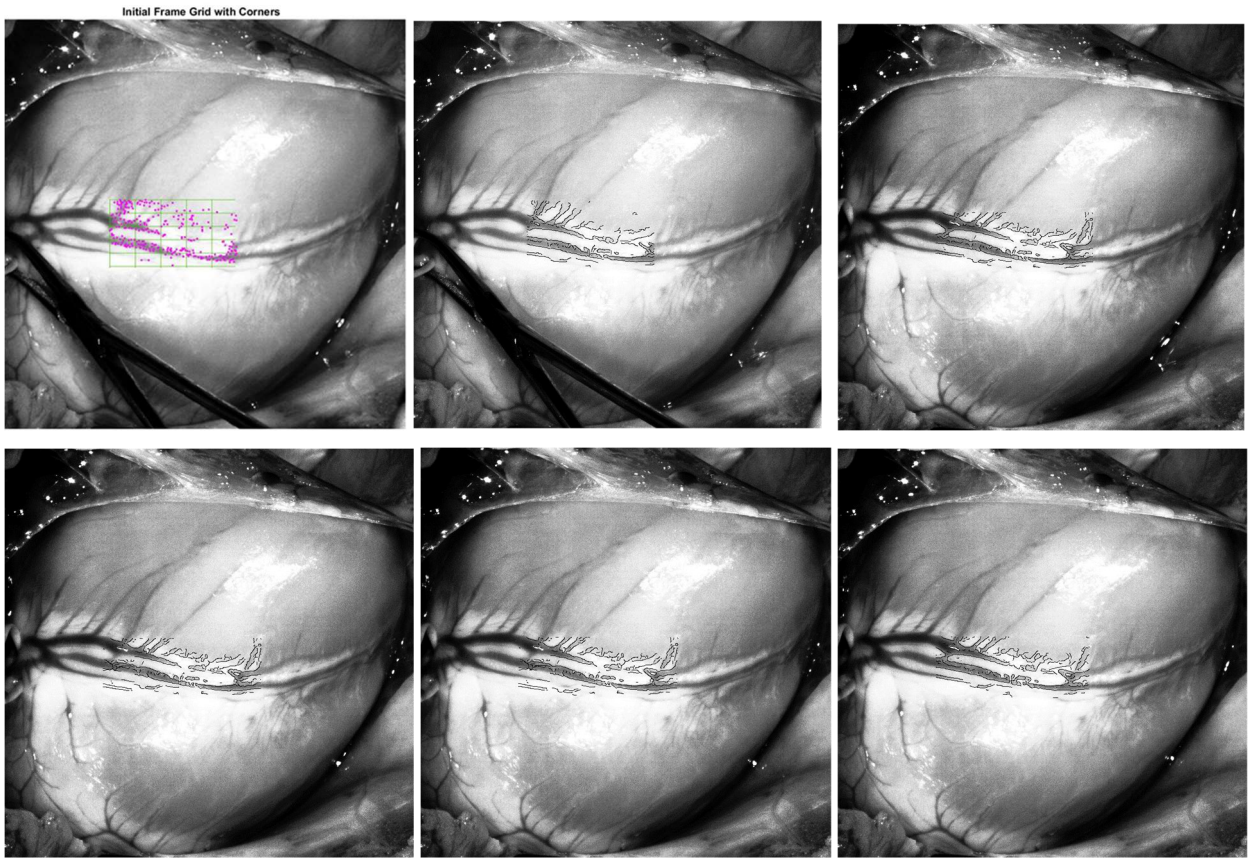


Figure 49: Data Set 4 Tracking Results Starting from Frame 1, 2, 2354 (top) to Frame 2355-2357 (bottom)

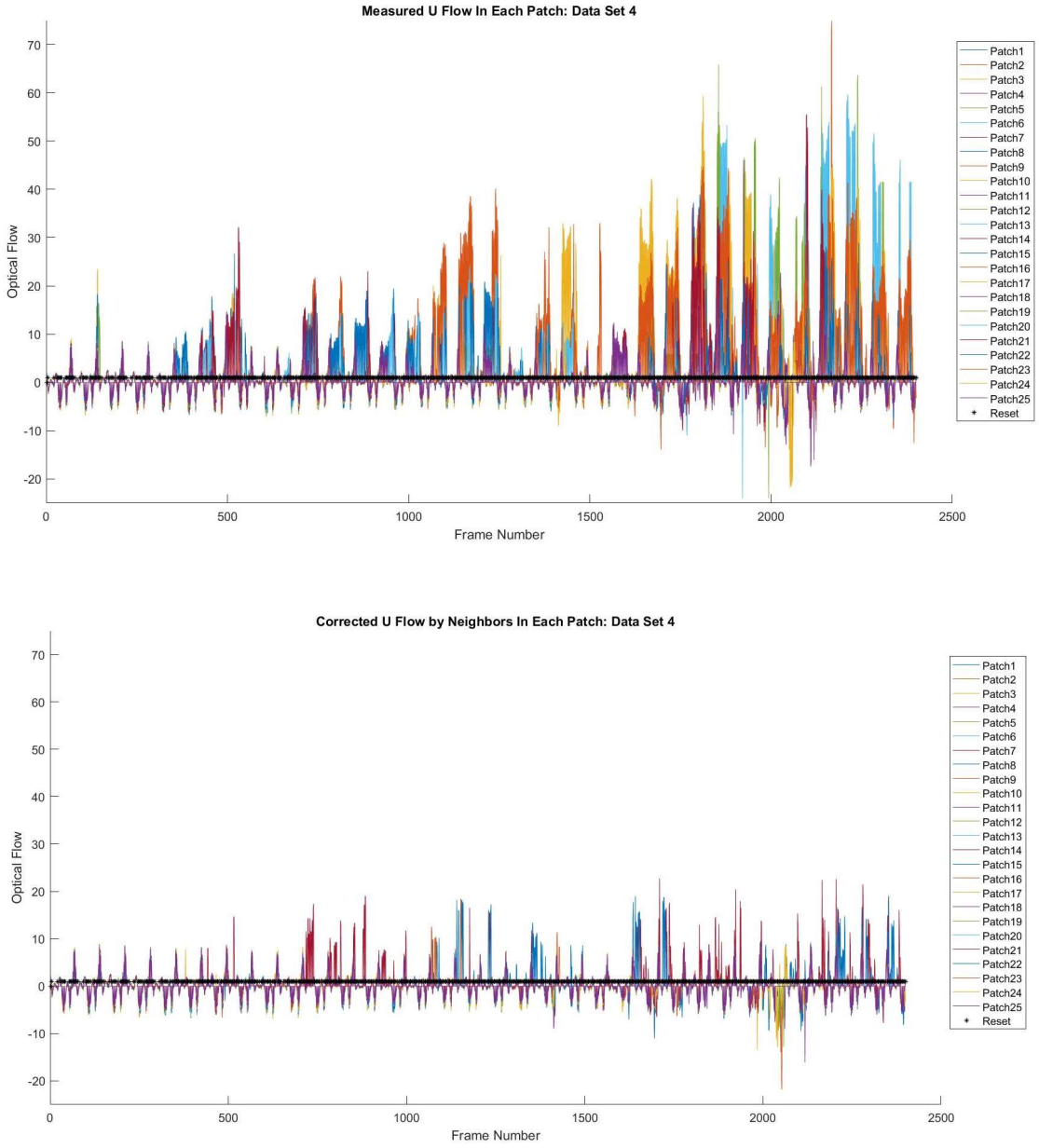


Figure 50: Measured U Optical Flow vs Corrected Flow by Neighbors

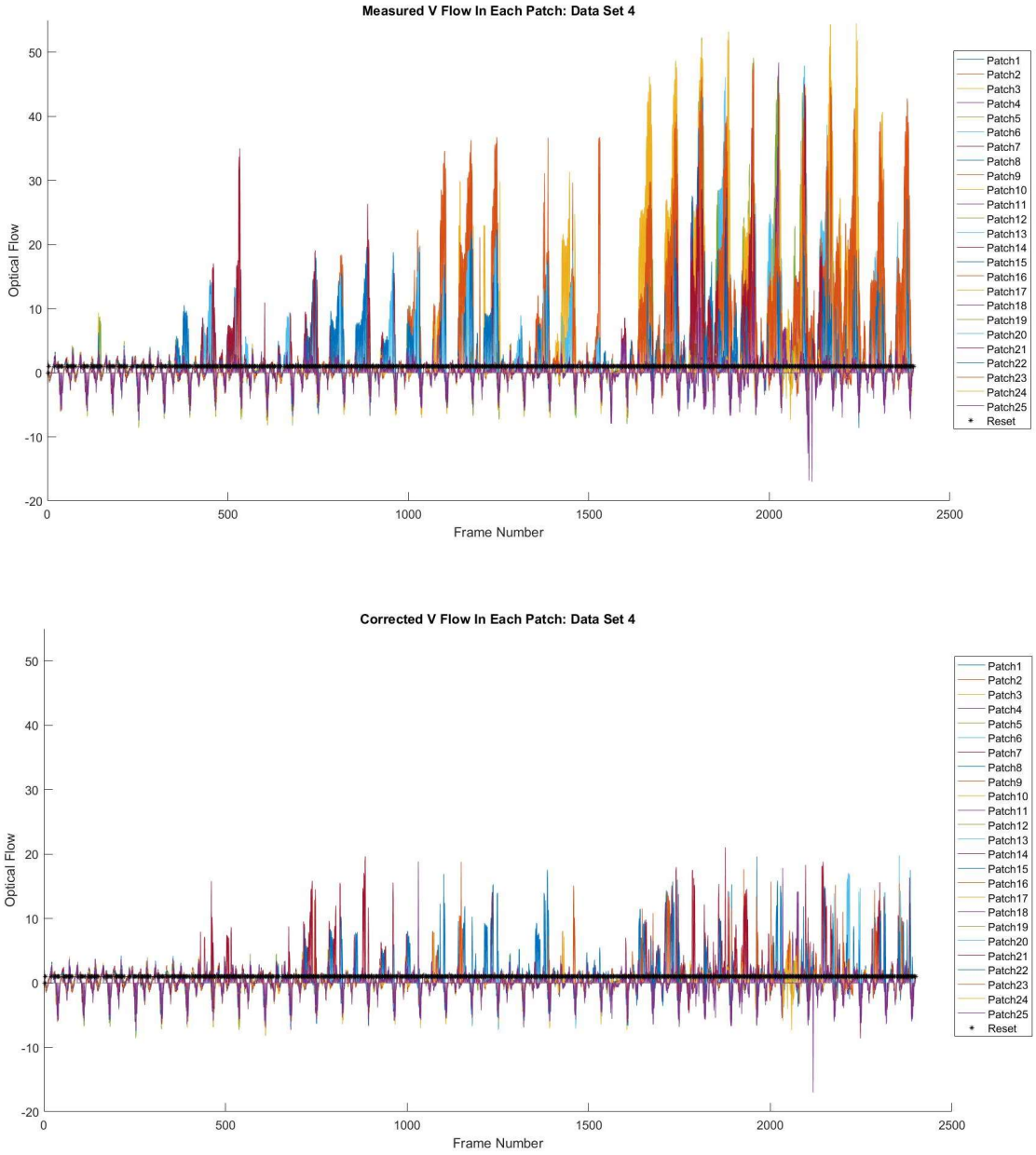


Figure 51: Measured V Optical Flow vs Corrected Flow by Neighbors

The last set of was successfully tracked over 2430 frames. Figure 52 below demonstrates tracking and reinitialization from frames 1, 2, and 2321-2324. These results show the middle and top left edges of the ROI fall off track and then realign by the last image frame. The total number of resets was found to be 286 times or 11.7% of the total number of frames. The average optical flow measure for each patch is also shown in Figures 53 and 54 below along with the corrected flow for each patch that does not pass the difference thresholds set. These results show that there

exists a distinct pattern to the movement of the ROI as the blood vessel always returns to its original position at the end of every heart cycle. A flagged frame is also highlighted by a black star to show when the optical flow one or more patches is considered an outlier that does not pass a piecewise constraint.

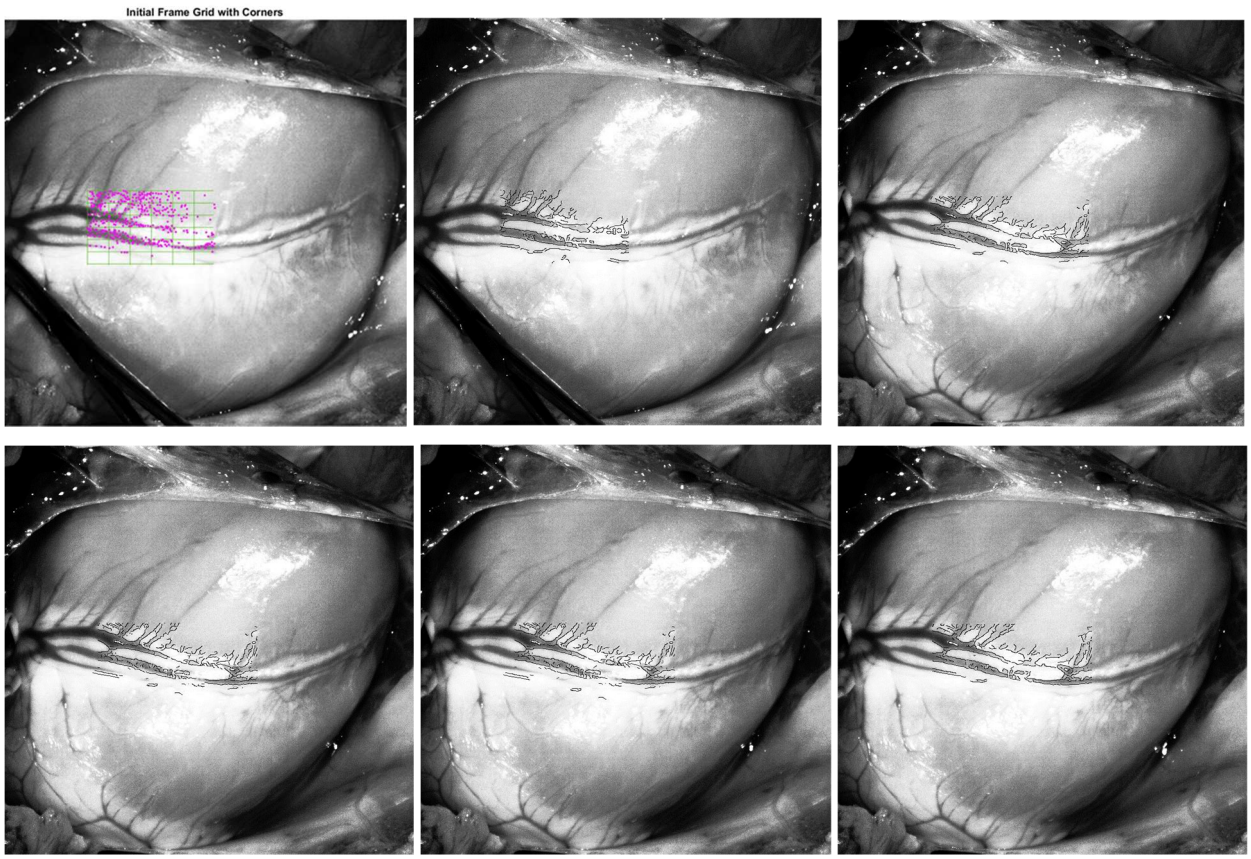


Figure 52: Data Set 5 Tracking Results Starting from Frame 1, 2, 2321 (top) to Frame 2322-2324 (bottom)

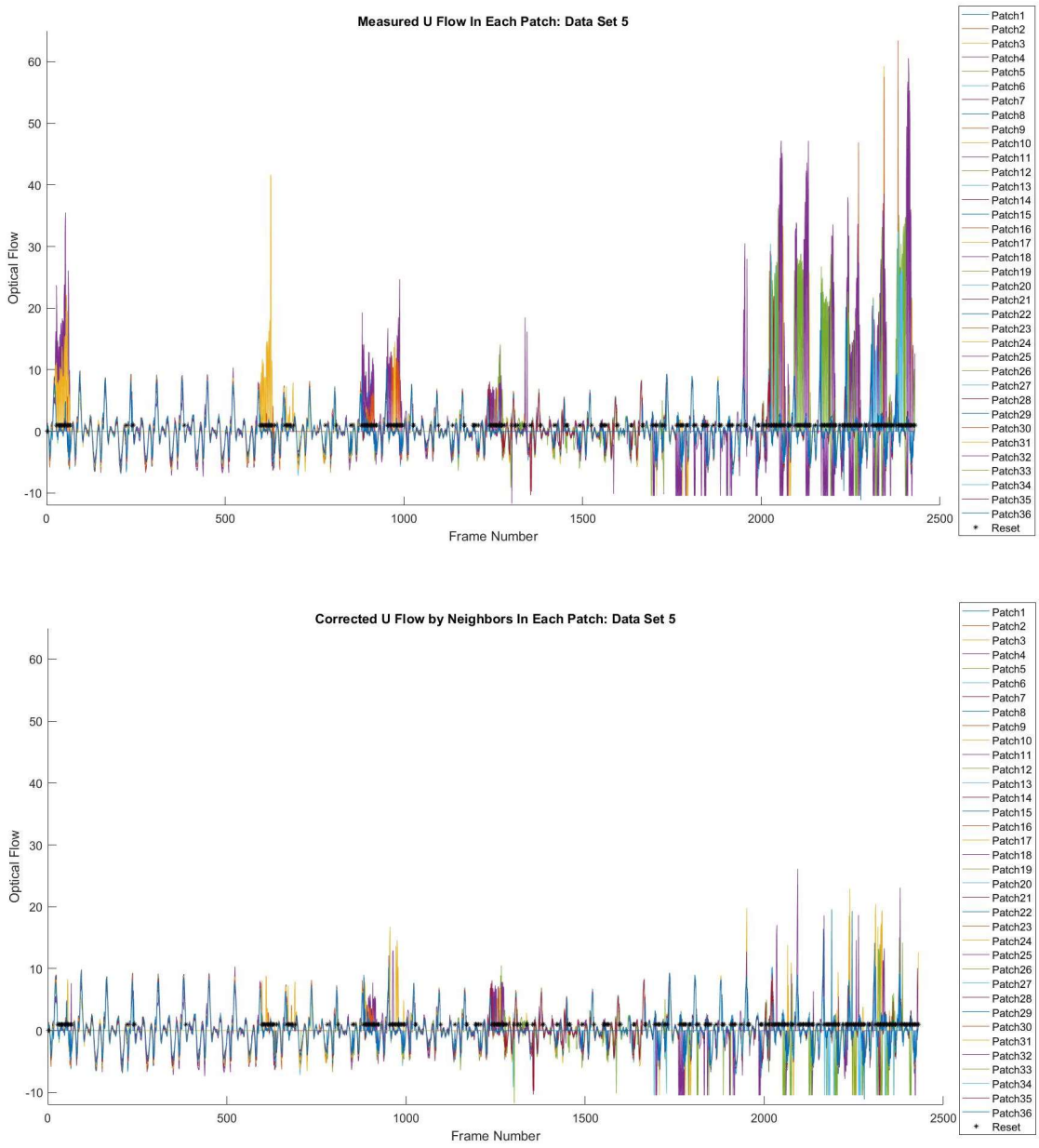


Figure 53: Measured U Optical Flow vs Corrected Flow by Neighbors

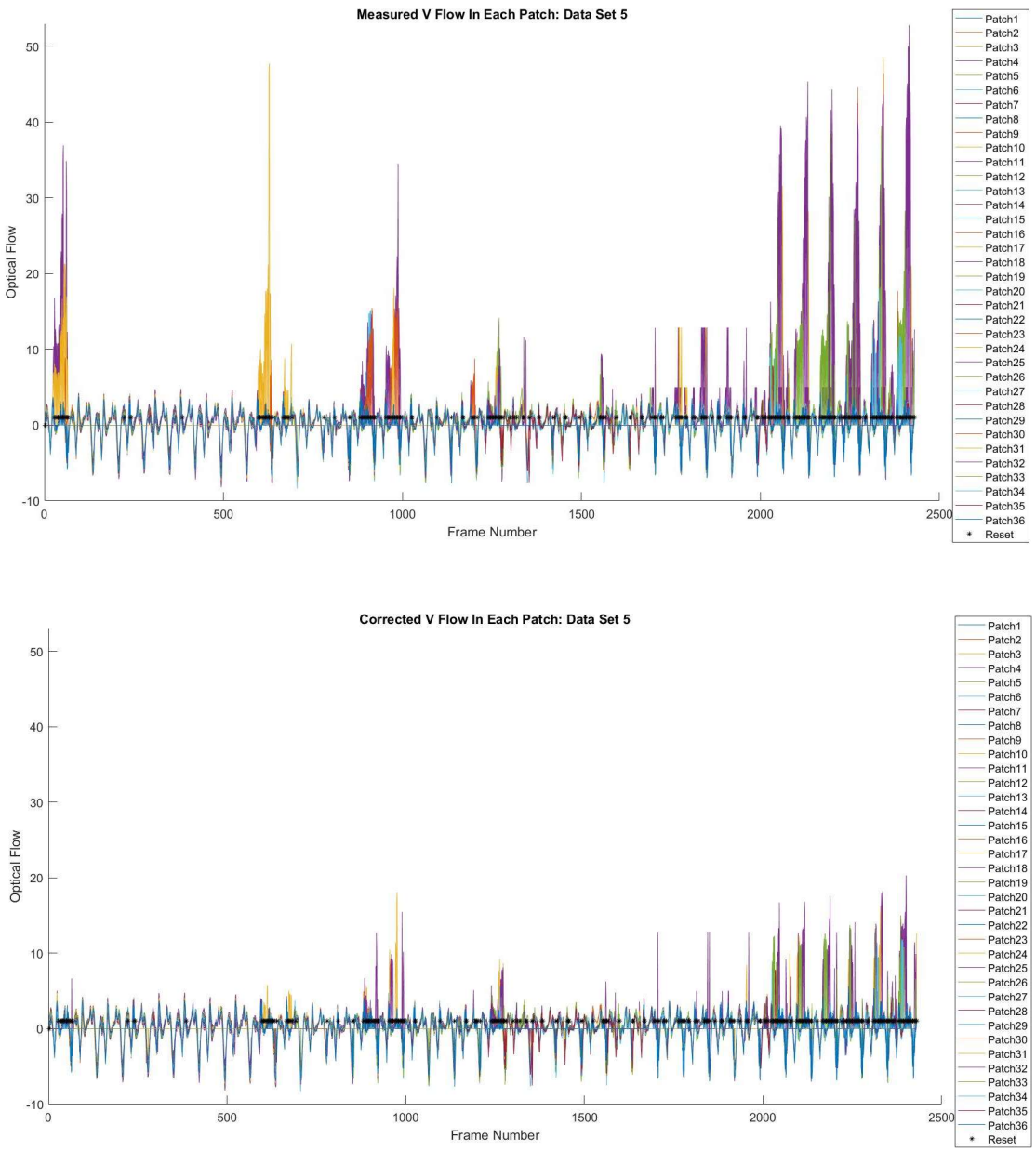


Figure 54: Measured V Optical Flow vs Corrected Flow by Neighbors

One last set of results to present includes the motion for each data set in the first few heart cycles. Figures 55, 56, 57, 58, and 59 demonstrate the optical flow measured with the KLT tracker in the U and V directions over five heart cycles. The results verify that a distinct pattern can be observed in each data set. In the corrected U and V flow measurements, we noticed that all the patches tend to follow similar patterns in general. However, their motion can differ by several pixels across the ROI. It is due to the non-rigid nature of the problem. A machine

learning approach could be added to generate a learned model that could make tracking predictions on future data based on a pattern over the first few heart cycles.

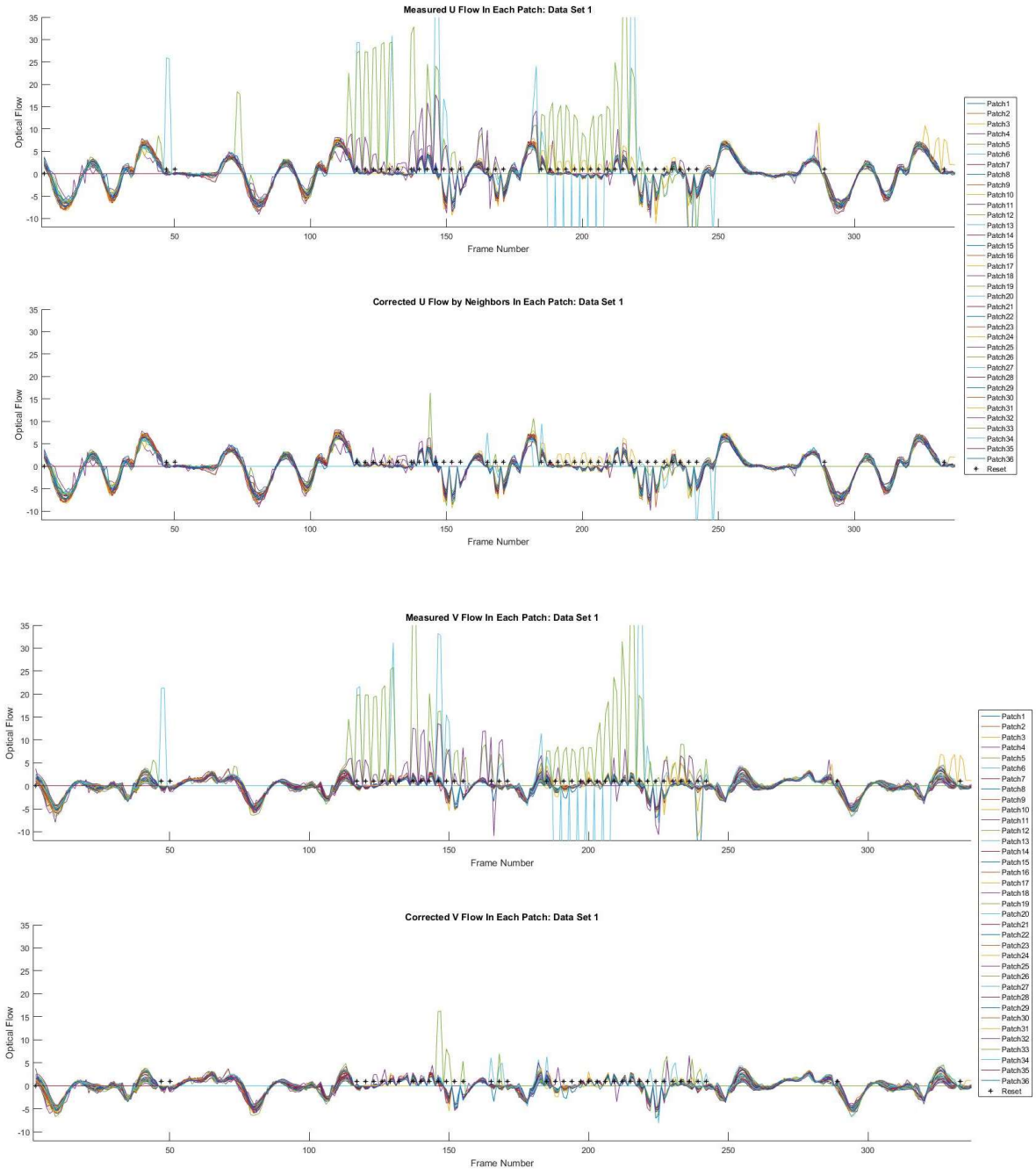


Figure 55: Measured and Corrected U and V Flow Over Five Heart Cycles – Data Set 1

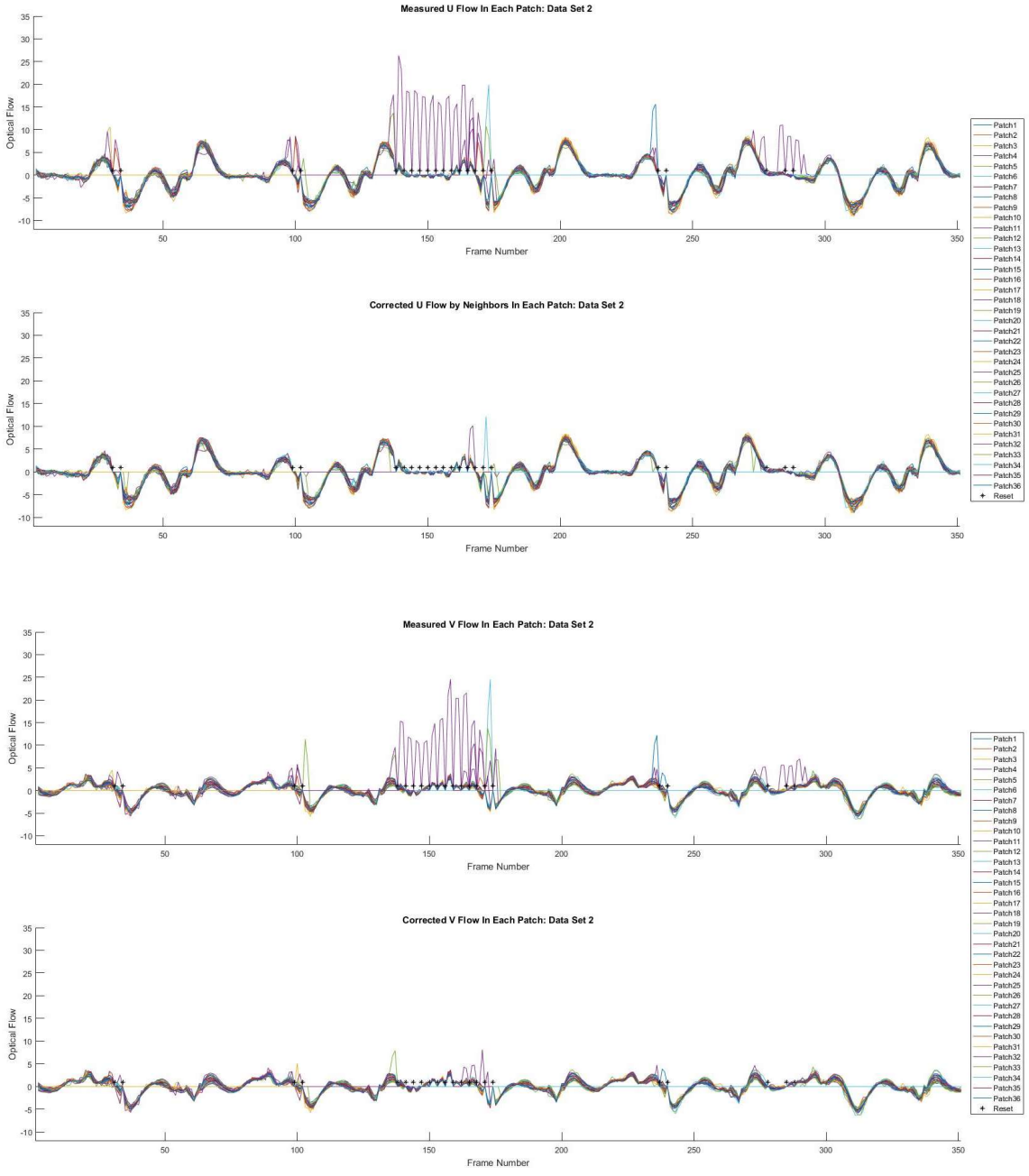


Figure 56: Measured and Corrected U and V Flow Over Five Heart Cycles – Data Set 2

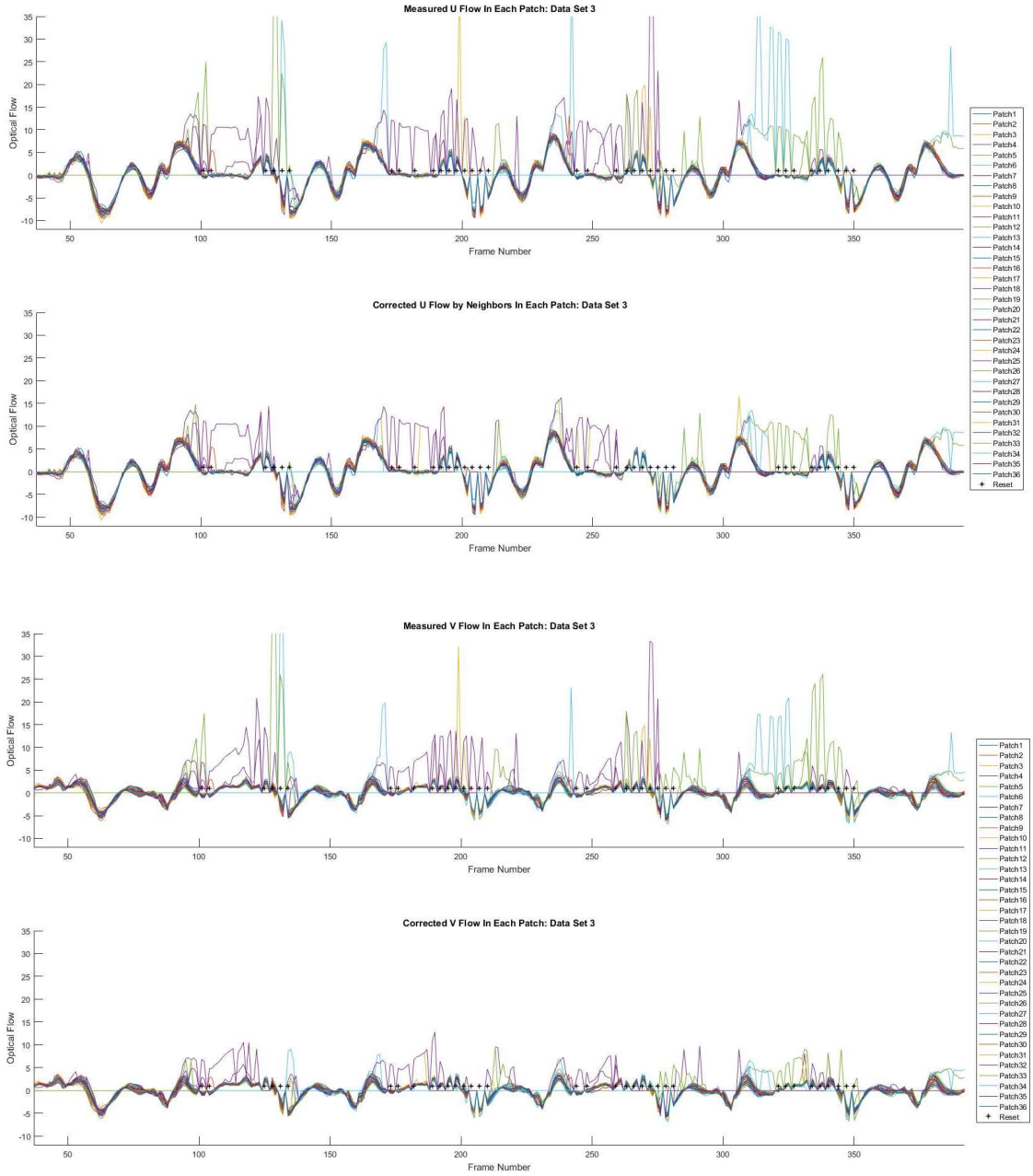


Figure 57: Measured and Corrected U and V Flow Over Five Heart Cycles – Data Set 3

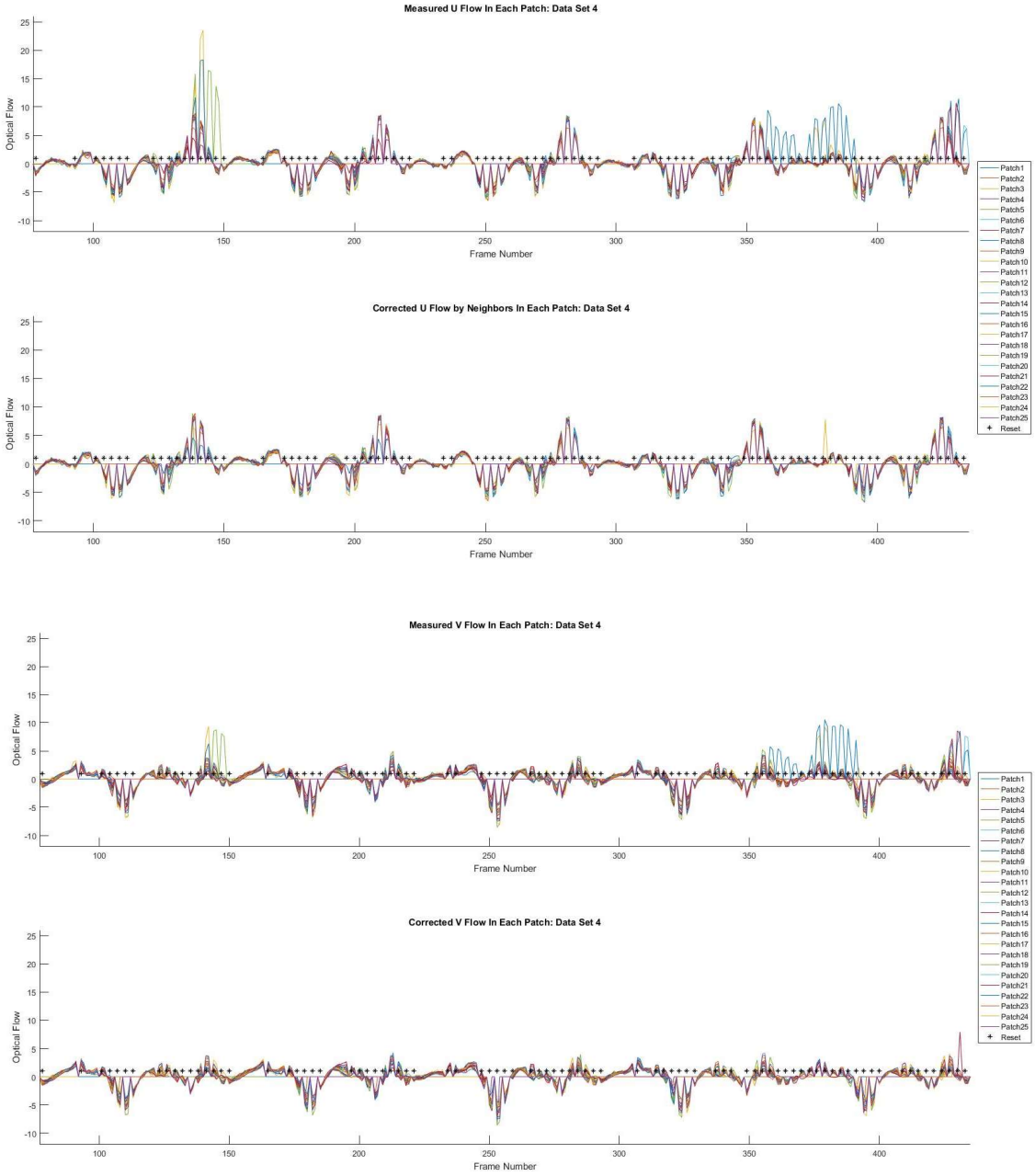


Figure 58: Measured and Corrected U and V Flow Over Five Heart Cycles – Data Set 4

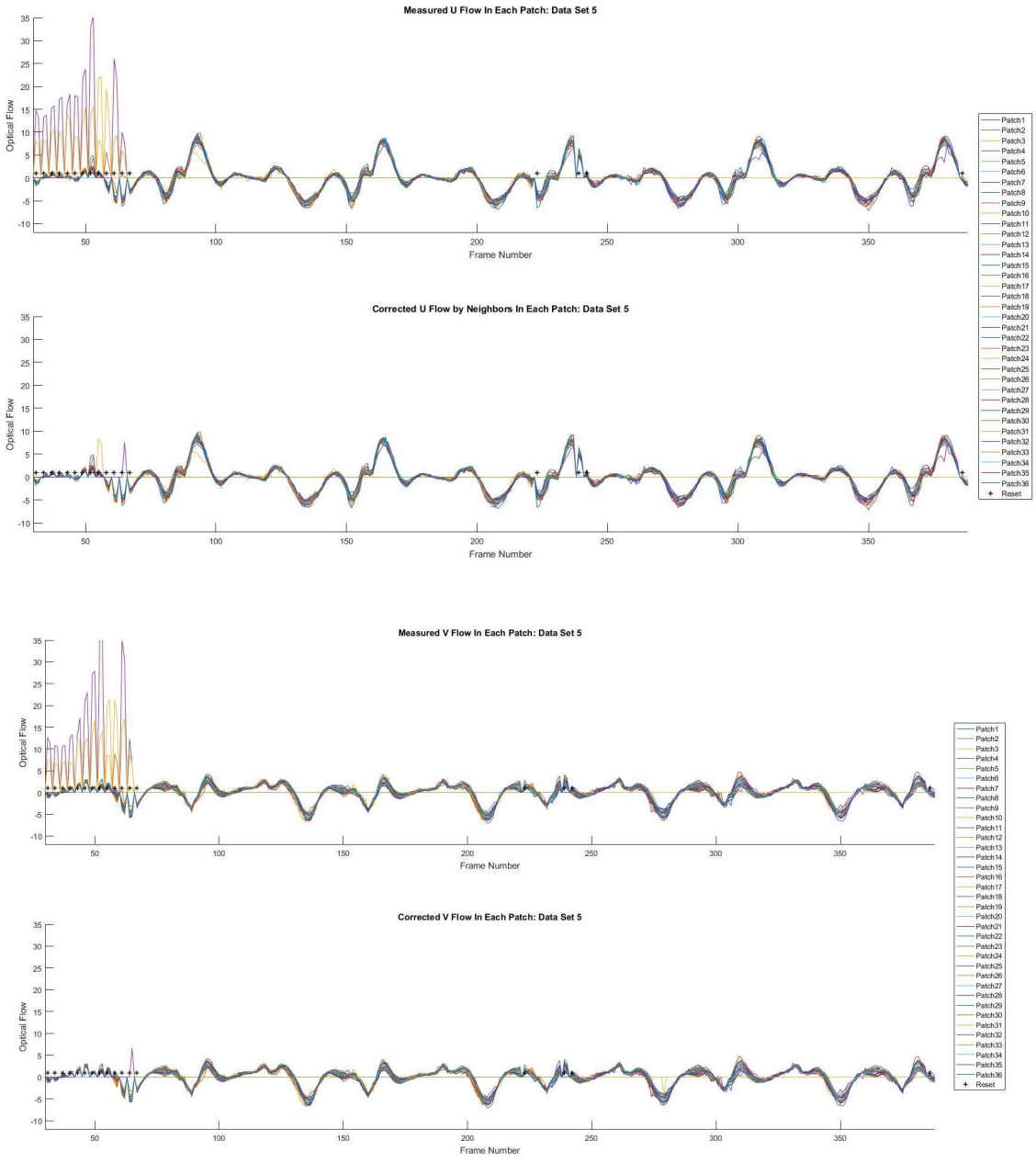


Figure 59: Measured and Corrected U and V Flow Over Five Heart Cycles – Data Set 5

CHAPTER 5: DISCUSSION

The results show that the piecewise tracking algorithm was able to break up a selected ROI into smaller patches that could be estimated as small pseudo-rigid segments. The algorithm automatically calculated the number of patches and motion thresholds needed to meet the piecewise geometrical constraints during the initialization step.

5.1: Initialization of Parameters

For each data set, the algorithm was initialized with a set of parameters, which are similar across data sets. The configuration of these parameters can contribute to the overall performance of tracking, which can be evaluated, for example, by the likelihood of reinitializations or resets needed. It was confirmed through visual inspection that 93% of the total resets in all five data sets are due to a loss of tracking, and only 7% of them were considered false alarms. It is reasonable to assess the tracking performance by monitoring the resets. Each reinitialization step that is counted as a false alarm occurs when the motion estimated does not pass the set thresholds, but does follow the actual motion of the ROI. Therefore, with these parameters, the incorrect motion estimates are mostly due to a loss of tracking as the estimated motion does not meet the set thresholds and does not follow the actual movement of the ROI. The two parameters that are found to be the most influential to the overall performance are the number of corners detected in the initial frame and number of patches used in tracking. Although both parameters are iteratively optimized in the automatic approach, it is of interest to consider their impact on tracking.

The number of corner features used to track each patch could have changed by varying the initial Harris corner detection threshold. 12,626 corners were detected by using an initial threshold of 0.005 in the current software implementation, which was the same value used across all data sets in this analysis. The number of patches was then determined by grouping some of these corner features into small patches, and testing different patch sizes to follow a normal distribution to meet the first constraint that every patch can be assumed to follow a pseudo-rigid motion. To test if the current performance of the tracking algorithm is the most efficient, the number of resets for one data set can be compared by varying these two parameters. Also since

the motion of every data set is non-rigid and of a biological heart, the same results shown are to be expected with the other four data sets.

The first parameter to be tested is the number of corners detected in the first frame, while keeping the patch number constant. For data set two, the number of corners was increased to 15,305, which corresponds to a Harris corner detection threshold of 0.001, and the number of resets was found to increase to 592 or 24% of the total image frames. The number of corners was also increased to the maximum 16,529 corners, which corresponds to a threshold of 0.0001, and the number of resets increased again to 648 or 26%. This test shows that if the number of corners initially detected is too large, it is more likely that the features detected are "bad" features that do not represent the location of a true corner. The increase in the number of "bad" features will lead to an incorrect motion estimation and cause the number of flagged frames and resets to increase. The number of corners was also lowered to test the algorithm performance. The corner detection threshold was first set to 0.008 to lower the number of corners detected to 10,331, and the number of resets was found to increase to 595 (24%). The number of corners was then set to a minimum of 9,007 corners, which corresponds to a threshold of 0.01, and the reset number was found to also increase to 566 (23%) times. This test shows that even though it is less likely to have a "bad" feature by lowering the number of corners detected, the number of corners per patch will decrease. The decrease in the number of corners per patch will lead to a higher chance of an outlier corner motion to cause an incorrect patch estimate and a flagged frame or reset. Overall, the number of corners initially detected at 12,626, with a threshold of 0.005, results in the lowest reset number at 540 resets or 22% to provide an optimal performance of the algorithm. It is also expected that either increasing or decreasing the number of corners used for every data set will lead to the same conclusion. These results show that the number of corners detected and overall tracking performance is not very sensitive to the actual threshold used, as the increase in the number of resets does not change by a significant amount. However, a step can be added to the initialization phase to determine an optimal threshold for the initial corner detection by testing for any outlier motion when this threshold is varied.

Next, the impact of the number of patches is investigated. The number of patches was increased to 49 and then increased again to 64 patches while keeping the corner detection threshold constant, to test if lowering the patch sizes and adding more patches will improve the

pseudo-rigid estimation of the ROI. Increasing the patch number to 49 does not change the number of resets significantly at 549 times (22.5%), but the number of corners per patch changes to 12. Changing the patch number to 64, however, does increase the number of resets to 664 times (27%) and the number of corners decreases per patch to 9. This increase in patch number decreases the patch size to lower the number of corners used in estimating the motion of each patch. The decrease in the number of corners per patch will lead to an inaccurate motion estimate that could be dominated by an outlier corner motion that will increase the number of flagged frames and resets. Also, if the number of patches were to decrease to allow for a larger patch size, each patch will still follow a non-rigid motion that could not be accurately estimated as an approximately rigid segment with optical flow. This test shows that the automatic initialization of a patch number to 36 patches, that all follow a pseudo-rigid motion within each patch, results in the optimal performance of the tracking algorithm. The same results are also expected for the other four data sets.

In addition to both parameters, the frame selected to start the initialization could influence the tracking performance of the algorithm. In the results shown in the previous section, the initial frame for every data set started during the relaxation or diastole phase of the heart cycle. This allowed the algorithm to start at the beginning of a heart cycle to improve the initialization of the piecewise grid and optical flow tracking. The initial frame was changed to show that starting the tracking process during the relaxation phase of the heart cycle provides a more accurate initialization and improves performance. To demonstrate the effect of the starting frame, the algorithm was changed for data set two to first start at frame 32, which corresponds to the middle of the heart cycle. The patch number selected by the algorithm was 36 and the algorithm was able to track the ROI throughout the entire data set, but the number of resets increased to 555 (23%). This test shows that the tracking algorithm can begin at the middle of the heart cycle, but to improve performance, it is better to begin tracking during the relaxation phase. The starting frame was then set to frame 55, which corresponds to the contraction phase of the heart cycle. The large motion change during this part of the heart cycle leads to a decrease in the patch number to 16, and the algorithm was unable to track the ROI throughout the entire data set. With a large patch size, the motion estimated for each patch was inaccurate as the patches still follow a non-rigid motion. Estimating the non-rigid motion with a rigid optical flow measure will lead to a loss or disconnect of edges in the ROI. By moving the starting frame away from the

relaxation phase of the heart cycle, the initialization and tracking results change to lower the performance of the algorithm developed. These results support keeping the initial frame constant during this phase for future heart motion tests.

5.2: Recovery from Loss of Track and Reinitialization

The algorithm was able to continuously track the ROI and recover from tracking failures for every image frame in all data sets, except for data set one. In this data set, the algorithm was unable to reinitialize due to a complete occlusion. Even though the algorithm is robust against temporary loss of data or occlusions, the complete occlusion changed the image data, and did not allow the ROI to be distinguished from the blockage. To properly handle this issue, the reinitialization step must be improved to allow for a temporary stop in the tracking process until the ROI is back in view. In a real-time implementation, this can be done with a user-controlled pause of the tracking process. However, this step could be automated by allowing the algorithm to detect when all patches were to not follow the piecewise constraints and stop tracking for the next few frames.

To summarize, the overall tracking performance of the algorithm presented was determined to be efficient. Changing the two most influential parameters, the number of corners detected and patch number/size, does affect this performance as the number of resets increased. However, selection of the initial frame appears to have a greater impact. The initial frame set at the relaxation phase of the heart cycle provides the best tracking performance, which requires an estimate of the heart cycle. The heart cycle can be measured with additional sensory, or directly observed in the imagery data. The motion estimated in each patch was also found to follow a pattern for the movement of the ROI in each heart cycle. In further studies, a machine learning approach could be added to generate a learned model that could make tracking predictions on future data specifically based on their individual pattern over the first few heart cycles. The learned model could also be used to provide a smart selection of the initial frame to begin initialization and tracking at the beginning of the heart cycle. The reinitialization step could also be improved from this model by determining where the discontinued frame is located in the current heart cycle, and find the image frame from the same location in the previous heart cycle. This step would improve the computation speed and realignment accuracy to reset the piecewise grid and restart the tracking process.

CHAPTER 6: CONCLUSION

To successfully track a non-rigid system, such as the biological heart, a piecewise tracking approach was developed to break up a selected ROI into small segments (patches) that are assumed to be approximately rigid. The number of pseudo-rigid segments was determined by following two constraints. Both constraints were defined based on smoothness of the physical model, and enforced on tracking residuals. Image tracking that does not conform to either constraint could be flagged as a failure and trigger a reinitialization step. The reinitialization step was able to realign the piecewise grid and reset the features within each patch to be tracked.

The results show that the proposed piecewise approach was indeed able to track a biological heart. Overall, it was able to meet all objectives outlined in the introduction as 1) it does not require an underlying motion model to be known, 2) it quantifies the current tracking estimate, and 3) it can recover from a tracking failure.

The tracking algorithm could be improved in future studies. The results showed that the algorithm is robust against temporary blockage or loss of images. However, with severe occlusion, for example, blockage over the entire ROI over a period of time, it may not be able to recover as successfully. The reinitialization step may be improved to allow for the tracking algorithm to stop running when a complete occlusion occurs, and restart when the ROI is back in view. In a real-time implementation, this can be done with a user-controlled pause of the tracking process. However, this step could be automated by allowing the algorithm to detect when all patches were to not follow the piecewise constraints and stop tracking for the next couple of frames.

The motion estimated in each patch was also found to follow a pattern for the movement of the ROI in each heart cycle. In further studies, a machine learning approach could be added to generate a learned model that could make tracking predictions on future data, allow for a smart selection of the initial frame, and improve reinitialization by matching any discontinued frame to its previous frame from the last heart cycle. This model will be specifically based on the individual pattern of a ROI over the first few heart cycles. This would still allow the tracking algorithm to run with a piecewise approach and not require an underlying motion model to be known before tracking process begins

Furthermore, the proposed approach could also be feasible for tracking other biological structures that exhibit a non-rigid motion model. Additional testing with images of a variety of biological systems will provide more insight on the non-parametric piecewise tracking process.

Finally, the approach was designed and developed using computationally efficient image processing methods. It would be feasible to implement and optimize the approach for a real-time imaging system.

REFERENCES

Arif et. al. (2014). Tracking Using Motion Estimation with Physically Motivated Inter-Region Constraints. *IEEE Transactions on Medical Imaging* , 1875-1889.

Axel, & Dougherty. (1989). Heart wall motion: Improved method of spatial modulation of magnetization for MR imaging. *Radiology*, 349-360.

Axel, L., & Dougherty, L. (1989). MR imaging of motion with spatial modulation of magnetization. *Radiology*, 841-845.

Bader et. al. (2007). Model-Based Motion Estimation of Elastic Surfaces for Minimally Invasive Cardiac Surgery. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2261-2266.

Bay et. al. (2007). SURF: Speeded Up Robust Features. *Computer Vision - EECV 2006 Lecture Notes in Computer Science*, 404-417.

Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 679-698.

Chandrashekara et. al. (2003). Construction of a Statistical Model for Cardiac Motion Analysis Using Nonrigid Image Registration. *Radiology*, 1-12.

Choi et. al. (2009). Performance Evaluation of the RANSAC Family.

Cremers, D., & Soatto, S. (2004). *Motion Competition: A Variational Approach to Piecewise Parametric Motion Segmentation.* London: Springer.

Cremers, D., & Wedel, A. (2011). Optical Flow Estimation. In *Stereo Scene Flow for 3D Motion Analysis* (pp. 5-34). London: Springer.

Fischer et. al. (1993). Improved Myocardial Tagging Contrast . *Magnetic Resonance Med*, 191-200.

Gimp. (2017). *Convolution Matrix: Generic Filters*. Retrieved from gimp.org: https://docs.gimp.org/en/plug-in-convmatrix.html

Hartley, R., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision.* Cambridge: Cambridge University Press.

Hassanein et. al. (2014). Performance of Optical Flow Tracking Approaches for Cardiac Motion Analysis. *2nd Midle East Conference on Biomedical Engineering*.

Hassner et. al. (2013). When Standard RANSAC is Not Enough. *Machine Vision and Applications* , 971-983.

Hilsenbeck et al., O. (2016). Software tools for single-cell tracking and quantification of cellular and molecular properties. *Nature Biotechnology*, 703-706.

Ibrahim, E.-S. H. (2011). Myocardial tagging by Cardiovascular Magnetic Resonance: evolution of techniques–pulse sequences, analysis algorithms, and applications. *Journal of Cardiovascular Magnetic Resonance* , 1-40.

Juan et. al. (2010). A Scene Matching Algorithm Based on SURF Feature. *IEEE*, 1-4.

Lucas, B., & Kanade , T. (1981). An Iterative Image Registration Technique. *Proceedings of Imaging Understanding Workshop* , 121-130.

MathWorks. (2017). *Histogram Equalization*. Retrieved from MathWorks: https://www.mathworks.com/help/images/histogram-equalization.html

Najarian, K., & Splinter, R. (2012). *Biomedical Signal and Image Processing.* Boca Raton, Florida: CRC Press.

Phillips, D. (2000). *Image Processing in C.* Lawrence, Kansas: R&D Publications.

Ren, X. (2008). Local Grouping for Optical Flow. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1-8.

Richa et. al. (2011). Towards Robust 3D Visual Tracking for Motion Compensation in Beating Heart Surgery. *Medical Image Analysis*, 302-315.

Shi, J., & Tomasi, C. (1994). Good Features to Track. *IEEE Conference Paper on Computer Vision and Pattern Recognition*, 1-8.

Tomasi, C., & Kanade, T. (1991). *Detection an Tracking of Point Features.*

Tuna et. al. (2013). Heart Motion Prediction Based on Adaptive Estimation Algorithms for Robotic-Assisted Beating Heart Surgery . *IEEE Transactions on Robotics*, 261-276.

Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *CVPR*, 511-518.

WHO. (2016, September). *Cardiovascular diseases (CVDs)*. (WHO) Retrieved April 28, 2017, from http://www.who.int/mediacentre/factsheets/fs317/en/

Zerhouni et al. (1988). Human heart: Tagging with MR imaging - a method for noninvasive assessment of myocardial motion. *Radiology*, 59-63.

Zhou, H., Yuan, Y., & Shi, C. (2008). Object Tracking using SIFT features and mean shift. *Computer Vision and Image Understanding*, 345-352.

APPENDIX: MATLAB ALGORITHM

The tracking algorithm developed in MATLAB will be attached in this section. The main function is to be attached first and is labeled as "NewRegionBreakdownTrackin5.m." This approach requires the use of many other functions, where most are included in the MATLAB package. Functions that are not part of that package, but used in this algorithm are also included to follow. These include "DeterminePatchNum4.m" and "DetermineBestFit2.m." The SURF and RANSAC functions are found in MATLAB but the functions used in this approach can be found in the following downloadable links as part of the MathWorks and GitHub community webpage.

SURF: https://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf--including-image-warp-

RANSAC: https://github.com/RANSAC/RANSAC-Toolbox

Main Algorithm:

```
%BT Thesis - Piecewise tracking algorithm loop for data set
close all; clc;
clear all;
%FLOW ON EDGES MOVE USING CORNERS AND KLT TRACKING FUNCTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Overall
%inputs:data set 800x800xN image,patch grid starting pts
%
%outputs:highlight edges in next image frame from opt flow result + prev.
%edge index pts and loop for all frames, write new images to file
%
%summary:the purpose of this program is to loop through an entire data set
%to track the boundary edges of a selected segment, fix failed estimate,
%and quantify current track estimate
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part1: Input data set and determine correct patch size and number for
%piecewise tracking and determine corners
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%data set 1
% load('ImgNIR_20160725T124609.mat');%breaks down 220-221
% ImgNIR=ImgNIR_20160725T124609;
% %frame num = 31 and only all pass at 15
% for q=1:25
%     I=histeq(ImgNIR(:,:,q));
%     ImgCycle(:,:,q)=I;
% end
%data set 2
% load('20160725T124323.mat');
% ImgNIR=IDS.ImgNIR;
% %frame num = 31
```

```
% for q=1:25
%     I=histeq(ImgNIR(:,:,q));
%     ImgCycle(:,:,q)=I;
% end
% data set 3
% load('20160725T124909.mat');%breaks:1210-1213
% ImgNIR=IDS.ImgNIR;
% %frame num = 25
% for q=1:25
%     I=histeq(ImgNIR(:,:,q));
%     ImgCycle(:,:,q)=I;
% end
% data set 4
% load('20160725T132840.mat');%holds w/ high motion like 1941-1942
% ImgNIR=IDS.ImgNIR;
% %frame num = 31
% for q=1:25
%     I=histeq(ImgNIR(:,:,q));
%     ImgCycle(:,:,q)=I;
% end
% data set 5
load('20160725T133113.mat');%breaks:
ImgNIR=IDS.ImgNIR;
%frame num = 26
for q=1:25
    I=histeq(ImgNIR(:,:,q));
    ImgCycle(:,:,q)=I;
end

%num frames
numframes=size(ImgNIR,3);

%create directory to save output images
mkdir('C:\ECU\Thesis\DataResultsSet5\TimedTest1');

%run Equalize function to apply histeq to img frame 1 and 2
img1=histeq(ImgNIR(:,:,1));
img2=histeq(ImgNIR(:,:,2));
count=1;
MinPatchCorners=0;

%detect corners across image
corners=detectMinEigenFeatures(img1,'MinQuality',0.005);

%have user select start and end point within region of interest
figure,imshow(img1), hold on;
title('Region of Interest Input');
%press return(enter) to stop selection
% [x,y]=ginput(2);

%1
% x=[227;475];
% y=[344;482];
% 2
% x=[231;480];
% y=[333;478];
```

```
%3
% x=[180;434];
% y=[327;460];
%4
% x=[221;465];
% y=[353;484];
%5
x=[169;415];
y=[337;481];

pt1=[x(1,1);y(1,1)];
pt2=[x(2,1);y(1,1)];
pt3=[x(1,1);y(2,1)];
pt4=[x(2,1);y(2,1)];
plot(pt1(1,1),pt1(2,1),'*','MarkerSize',8);
plot(pt2(1,1),pt2(2,1),'*','MarkerSize',8);
plot(pt3(1,1),pt3(2,1),'*','MarkerSize',8);
plot(pt4(1,1),pt4(2,1),'*','MarkerSize',8);
StartPt=pt1;
EndPt=pt4;

%minimize amount of corners to within area near selected region
ind=0;
for w = 1:length(corners.Location(:,1));
    if round(corners.Location(w,2)) >= (StartPt(2,1)-20) &&
round(corners.Location(w,2)) <= (EndPt(2,1)+20) ...
            && round(corners.Location(w,1)) >= (StartPt(1,1)-20) &&
round(corners.Location(w,1)) <= (EndPt(1,1)+20)
        ind=ind+1;
        ReducedCorners(ind,:)=[corners.Location(w,1) corners.Location(w,2)
w];
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part 2: Run function to automatically determine patch number between
%frames 1-2 that pass motion criteria of normally dist within patch and
%continuous across neighbors. Then store edges+corners within each Patch of
%img1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function to determine correct patch number
% patchnum=DeterminePatchNum(img1,img2,StartPt,EndPt,ReducedCorners,corners);
% [FinalPatchNum,UThresh,VThresh] =
DeterminePatchNum2(img1,ImgCycle,StartPt,EndPt,ReducedCorners,corners);
[FinalPatchNum,MaxUHorizThresh,MaxVHorizThresh,MaxUVertThresh,MaxVVertThresh,
NumCorners] =
DeterminePatchNum4(img1,ImgCycle,StartPt,EndPt,ReducedCorners,corners,MinPatc
hCorners);
%create cell array with N rows for each patch and 3 columns for left corner
%pt,corner index,edges index within each patch that is updated every
iteration
PatchInfo={};
patchnum=FinalPatchNum;
rowcount=sqrt(patchnum);%initial row count
diffY=abs(EndPt(2,1)-StartPt(2,1));
diffX=abs(EndPt(1,1)-StartPt(1,1));
```

```
sizeY=round(diffY/rowcount);
sizeX=round(diffX/rowcount);
if(mod(sizeX,2)>0)
    sizeX=sizeX+1;
end
if(mod(sizeY,2)>0)
    sizeY=sizeY+1;
end

% patchID=zeros(patchnum,1);
numedges=zeros(patchnum,1);

%apply blur + edge detection on region
I1Blur1=imgaussfilt(img1,0.5);
I1Blur1Edges=edge(I1Blur1,'canny',[0.035 0.0675]);

patchcount=0;
for Ycount = 0:(rowcount-1)
    StartY=round(StartPt(2,1)+(sizeY*Ycount));
    for Xcount = 0:(rowcount-1)
        patchcount=patchcount+1;
        index2=0;
        StartX=round(StartPt(1,1)+(sizeX*Xcount));
        StartCornerPt=[StartY,StartX];
        PatchInfo(patchcount,1)={StartCornerPt};

patchCorners=SaveCornerIndex(ReducedCorners,StartX,StartY,sizeX,sizeY);
        PatchInfo(patchcount,2)={patchCorners};
        patchOutline=I1Blur1Edges(StartY:((StartY+sizeY)-
1),StartX:((StartX+sizeX)-1));
        patchEdges=SaveEdgeIndex(patchOutline,StartX,StartY);
        EdgeNum(patchcount,1)=length(patchEdges(:,1));
        PatchInfo(patchcount,3)={patchEdges};
    end
end

    figure,imshow(img1),hold on;
    %make patch grid
    for k = StartPt(2,1):sizeY:(EndPt(2,1)+2)
        xgrid = [StartPt(1,1) (EndPt(1,1)+2)];
        ygrid = [k k];
        plot(xgrid,ygrid,'Color','g','LineStyle','-');
        plot(xgrid,ygrid,'Color','r','LineStyle',':');
    end

    for k = StartPt(1,1):sizeX:(EndPt(1,1)+2)
        xgrid = [k k];
        ygrid = [StartPt(2,1) (EndPt(2,1)+2)];
        plot(xgrid,ygrid,'Color','g','LineStyle','-');
        plot(xgrid,ygrid,'Color','r','LineStyle',':');
    end

    for mm=1:patchnum
        Corners=cell2mat(PatchInfo(mm,2));
        for m = 1:length(Corners(:,1))
            PlotCorners=round(Corners(m,:));
            plot(PlotCorners(:,1),PlotCorners(:,2),'.m');
```

```
        end
    end
    title('Initial Frame Grid with Corners');

%initialize tracker
tracker=vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker,corners.Location,img1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part3: Main Loop
%inputs:opt flow object and current 2 frames, init edge pts from frame 1
%outputs:next image frame with highlighted edges and resampled corners
%after grid start,stop is moved by AVG Flow in X and Y
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%init flags
Flag=0;
numflags=0;
numresets=0;
FlowAvg=struct;
FlowAvgCorrected=struct;
FlagIndex=zeros(numframes,1);
tic;
BeginTime=datestr(now)
while count < numframes
    UpdateTime=datestr(now);
    %increment current frame count,get prev(I1) and curr(I2) images
    count = count + 1;
    FlowStatus=1;
    I1=histeq(ImgNIR(:,:,count-1));
    I2=histeq(ImgNIR(:,:,count));

    %move corners using KLT tracker
    [Corners2Loc,validity,scores]=step(tracker,I2);

    %calc U and V flow with Average
    %save flow in each patch + calculate average flow in X and Y
    VFlowAVG = zeros(patchnum,1);
    UFlowAVG = zeros(patchnum,1);
    CornerFlow=zeros(patchnum,1);
    PatchCornerFlowU=struct;
    PatchCornerFlowV=struct;
    NoCorners=zeros(patchnum,1);
    ind7=0;
    for k = 1:patchnum
        CurrCorners=cell2mat(PatchInfo(k,2));
        Test=CurrCorners(1,3);
        if Test == 0%if no corners, do not track patch
            ind7=ind7+1;
            NoCorners(ind7,1)=k;
            PatchCornerFlowU(k).U=0;
            PatchCornerFlowV(k).V=0;
            UFlowAVG(k,1)=0;
            VFlowAVG(k,1)=0;
        else
            numcorners=length(CurrCorners(:,1));
            PatchU=zeros(numcorners,1);
```

```
        PatchV=zeros(numcorners,1);
        for kk=1:numcorners
            MovedIndex=CurrCorners(kk,3);
            PatchU(kk)=Corners2Loc(MovedIndex,1)-CurrCorners(kk,1);
            PatchV(kk)=Corners2Loc(MovedIndex,2)-CurrCorners(kk,2);
        end
        UFlowAVG(k,1)=mean(PatchU);
        VFlowAVG(k,1)=mean(PatchV);
        PatchCornerFlowU(k).U=PatchU;
        PatchCornerFlowV(k).V=PatchV;
    end
end
%save positive or negative flow values and save neighbor flow if flow
%in X or Y is = 0
for k=1:patchnum
    U=PatchCornerFlowU(k).U;
    V=PatchCornerFlowV(k).V;
        for kk = 1:length(U)
            if U(kk)==0
                ind1=kk;
                if ind1 > 1
                    ind1=ind1-1;
                else if ind1 == 1
                    for w=2:length(U)
                        if U(w) ~= 0
                            ind1=w;
                        end
                    end
                else
                    ind1=length(U);
                    end
                end
                U(kk)=U(ind1);
            end
        end
        for kk = 1:length(V)
            if V(kk)==0
                ind2=kk;
                if ind2 > 1
                    ind2=ind2-1;
                else if ind2 == 1
                    for w=2:length(V)
                        if V(w) ~= 0
                            ind2=w;
                        end
                    end
                else
                    ind2=length(V);
                    end
                end
                V(kk)=V(ind2);
            end
        end
        PatchCornerFlowU(k).U=U;
        PatchCornerFlowV(k).V=V;
        %calculate new avg flow in x and y
        UFlowAVG(k,:)=mean(PatchCornerFlowU(k).U);
```

```
            VFlowAVG(k,:)=mean(PatchCornerFlowV(k).V);
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Part 4: Check for Normal distribution within each patch by counting
    %outliers and Chi-square GOF test
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %run chi goodness of fit test to check if each flow is normal dist.
    UFlowChiTest=zeros(patchnum,1);
    VFlowChiTest=zeros(patchnum,1);
    UDistOutliers=0;
    VDistOutliers=0;
    for aa=1:patchnum
        if UFlowAVG(aa,1) ~= 0
            U=PatchCornerFlowU(aa).U;
            UFlowChiTest(aa,1)=chi2gof(U);
            if UFlowChiTest(aa,1)==1
                UDistOutliers=UDistOutliers+1;
            end
        end
        if VFlowAVG(aa,1) ~= 0
            V=PatchCornerFlowV(aa).V;
            VFlowChiTest(aa,1)=chi2gof(V);
            if VFlowChiTest(aa,1)==1
                VDistOutliers=VDistOutliers+1;
            end
        end
    end

    if UDistOutliers > 1 || VDistOutliers > 1
        Flag = 1;
    end

    %also count outliers outside of distribution and if #outliers > 5%
distribution is not normal and replace with average
    %for within patch, develop standard deviation and check for outliers
    %outside 2 sigma in both directions
    %develop standard deviation
    UFlowDEV=zeros(patchnum,1);
    VFlowDEV=zeros(patchnum,1);
    Udiffsum=0;
    Vdiffsum=0;
    for m = 1:patchnum
        CurrU=PatchCornerFlowU(m).U;
        CurrV=PatchCornerFlowV(m).V;
        for n=1:length(CurrU);
            Udiff=(CurrU(n,1)-UFlowAVG(m,1))^2;
            Vdiff=(CurrV(n,1)-VFlowAVG(m,1))^2;
            Udiffsum=Udiffsum+Udiff;
            Vdiffsum=Vdiffsum+Vdiff;
        end
        CurrUDev=sqrt(Udiffsum/length(CurrU));
        CurrVDev=sqrt(Vdiffsum/length(CurrV));
        UFlowDEV(m,1)=CurrUDev;
        VFlowDEV(m,1)=CurrVDev;
```

```
        Udiffsum=0;
        Vdiffsum=0;
    end
    %check for outliers outside this deviation in X and Y (U/V)
    PatchVariability=struct;
    OutliersU=zeros(patchnum,1);
    OutliersV=zeros(patchnum,1);
    RatioU=zeros(patchnum,1);
    RatioV=zeros(patchnum,1);
    for p=1:patchnum
        numoutliersU=0;
        numoutliersV=0;
        CurrU=PatchCornerFlowU(p).U;
        CurrV=PatchCornerFlowV(p).V;
        UIndex=ones(length(CurrU),1);
        VIndex=ones(length(CurrV),1);
        UAbove=UFlowAVG(p,1)+(2*UFlowDEV(p,1));
        UBelow=UFlowAVG(p,1)-(2*UFlowDEV(p,1));
        VAbove=VFlowAVG(p,1)+(2*VFlowDEV(p,1));
        VBelow=VFlowAVG(p,1)-(2*VFlowDEV(p,1));
        for q=1:length(CurrU)
            if CurrU(q) < UBelow
               %label as outlier
               numoutliersU=numoutliersU+1;
               UIndex(q)=0;
            end
            if CurrV(q) < VBelow
               %label as outlier
               numoutliersV=numoutliersV+1;
               VIndex(q)=0;
            end
            if CurrU(q) > UAbove
               %label as outlier
               numoutliersU=numoutliersU+1;
               UIndex(q)=0;
            end
            if CurrV(q) > VAbove
               %label as outlier
               numoutliersV=numoutliersV+1;
               VIndex(q)=0;
            end
        end
        %setup test 1=inside distribution,0=fail outside mean +/- 2 sigma
        PatchVariability(p).U=UIndex;
        PatchVariability(p).V=VIndex;
        OutliersU(p,1)=numoutliersU;
        OutliersV(p,1)=numoutliersV;
    end

    %set to not move points that are outliers or move by average flow in X/Y
    for p = 1:patchnum
        UIndex=PatchVariability(p).U;
        VIndex=PatchVariability(p).V;
%         if RatioU(p,1) > 5 && OutliersU(p,1) > 1
%     %         Flag=1;
%         else
            CurrU=PatchCornerFlowU(p).U;
```

```
            for q = 1:length(UIndex)
                if UIndex(q) == 0
                    CurrU(q)=UFlowAVG(p,1);
                end
            end
%         end
%         if RatioV(p,1) > 5 && OutliersV(p,1) > 1
%     %         Flag=1;
%         else
            CurrV=PatchCornerFlowV(p).V;
            for q = 1:length(VIndex)
                if VIndex(q) == 0
                    CurrV(q)=VFlowAVG(p,1);
                end
            end
%         end
        PatchCornerFlowU(p).U=CurrU;
        PatchCornerFlowV(p).V=CurrV;
        UFlowAVG(p,1)=mean(CurrU);
        VFlowAVG(p,1)=mean(CurrV);
    end

    %loop to track patches that do not move by neighbors flow
    for k=2:(patchnum-1)
        if UFlowAVG(k,1) == 0
            CurrCorners=cell2mat(PatchInfo(k,2));
            if length(CurrCorners(:,1)) >= 1
                if UFlowAVG(k-1) ~= 0
                    UFlowAVG(k,1)=UFlowAVG(k-1,1);
                else if UFlowAVG(k+1) ~= 0
                        UFlowAVG(k,1)=UFlowAVG(k+1,1);
                    end
                end
            end
        end
         if VFlowAVG(k,1) == 0
            CurrCorners=cell2mat(PatchInfo(k,2));
            if length(CurrCorners(:,1)) >= 1
                if VFlowAVG(k-1) ~= 0
                    VFlowAVG(k,1)=VFlowAVG(k-1,1);
                else if VFlowAVG(k+1) ~= 0
                        VFlowAVG(k,1)=VFlowAVG(k+1,1);
                    end
                end
            end
        end
    end

    %loop to not track patches with low edge count
    for k=1:patchnum
        if EdgeNum(k,1) < 10
            UFlowAVG(k,1)=0;
            VFlowAVG(k,1)=0;
        end
    end

    %store flow values
```

```
FlowAvg(count,1).U=UFlowAVG;
FlowAvg(count,1).V=VFlowAVG;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Part 5: Check for continuous flow between neighbors in Horizontal and
%vertical direction using thresholds from patchnum function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%now check variability across patches(compare averages)
%compare both U and V directions, thresh chosen in init function TTEST to
%have 0 or patches have same distribution
UPatchOutliersH=0;
UIndex2=ones(patchnum,1);
for k=1:rowcount
    if k == 1
        CurrRow=1;
    else
        CurrRow=(k*rowcount)-(rowcount-1);
    end
    for kk=CurrRow:(CurrRow+(rowcount-2))
       if UFlowAVG(kk,1)==0 || UFlowAVG(kk+1,1)==0
           UIndex2(kk)=1;
       else
           if abs(UFlowAVG(kk,1)-UFlowAVG(kk+1,1)) >= MaxUHorizThresh;
                UPatchOutliersH=UPatchOutliersH+1;
                UIndex2(kk)=0;
                Flag=1;
                fprintf('Flag between %d and %d U\n',kk,kk+1);
           end
       end
    end
end

VPatchOutliersH=0;
VIndex2=ones(patchnum,1);
for k=1:rowcount
    if k == 1
        CurrRow=1;
    else
        CurrRow=(k*rowcount)-(rowcount-1);
    end
    for kk=CurrRow:(CurrRow+(rowcount-2))
        if VFlowAVG(kk,1)==0 || VFlowAVG(kk+1,1)==0
            VIndex2(kk)=1;
        else
            if abs(VFlowAVG(kk,1)-VFlowAVG(kk+1,1)) >= MaxVHorizThresh
                VPatchOutliersH=VPatchOutliersH+1;
                VIndex2(kk)=0;
                Flag=1;
                fprintf('Flag between %d and %d V\n',kk,kk+1);
            end
        end
    end
end

%check variability across patches vertically
%compare both U and V directions
UPatchOutliersV=0;
UIndex3=ones(patchnum,1);
```

```
for k=1:rowcount
    CurrCol=k;
    for kk=CurrCol:rowcount:((rowcount-2)*rowcount)+CurrCol
        if UFlowAVG(kk,1)==0 || UFlowAVG(kk+rowcount,1)==0
            UIndex3(kk)=1;
        else
            UPatchDiff2=abs(UFlowAVG(kk,1)-UFlowAVG(kk+rowcount,1));
            if UPatchDiff2 >= MaxUVertThresh;
                UPatchOutliersV=UPatchOutliersV+1;
                UIndex3(kk)=0;
                Flag=1;
                fprintf('Flag between %d and %d U\n',kk,kk+rowcount);
            end
        end
    end
end

VPatchOutliersV=0;
VIndex3=ones(patchnum,1);
for k=1:rowcount
    CurrCol=k;
    for kk=CurrCol:rowcount:((rowcount-2)*rowcount)+CurrCol
        if VFlowAVG(kk,1)==0 || VFlowAVG(kk+rowcount,1)==0
            VIndex3(kk)=1;
        else
            VPatchDiff2=abs(VFlowAVG(kk,1)-VFlowAVG(kk+rowcount,1));
           if VPatchDiff2 >= MaxVVertThresh;
                VPatchOutliersV=VPatchOutliersV+1;
                VIndex3(kk)=0;
                Flag=1;
                fprintf('Flag between %d and %d V\n',kk,kk+rowcount);
           end
        end
    end
end

%velocity to neighbors
for p = 2:(patchnum-1)

     if UIndex2(p,1)==0
        if UFlowAVG(p,1) > 0 && UFlowAVG(p+1) > 0
            if UFlowAVG(p,1) > UFlowAVG(p+1,1)
                UFlowAVG(p,1)=UFlowAVG(p+1,1);
            else if UFlowAVG(p+1,1) > UFlowAVG(p,1)
                    UFlowAVG(p+1,1)=UFlowAVG(p,1);
                end
            end
        else if UFlowAVG(p,1) < 0 && UFlowAVG(p+1,1) < 0
             if UFlowAVG(p,1) < UFlowAVG(p+1,1)
                UFlowAVG(p,1)=UFlowAVG(p+1,1);
            else if UFlowAVG(p+1,1) < UFlowAVG(p,1)
                    UFlowAVG(p+1,1)=UFlowAVG(p,1);
                end
             end
        else
            if UFlowAVG(p,1) > UFlowAVG(p+1,1)
                UFlowAVG(p,1)=UFlowAVG(p+1,1);
```

```
            else if UFlowAVG(p+1,1) > UFlowAVG(p,1)
                    UFlowAVG(p+1,1)=UFlowAVG(p,1);
                end
            end

            end
        end
     end

    if VIndex2(p,1)==0
        if VFlowAVG(p,1) > 0 && VFlowAVG(p+1,1) > 0
            if VFlowAVG(p,1) > VFlowAVG(p+1,1)
                VFlowAVG(p,1)=VFlowAVG(p+1,1);
            else if VFlowAVG(p+1,1) > VFlowAVG(p,1)
                    VFlowAVG(p+1,1)=VFlowAVG(p,1);
                end
            end
        else if VFlowAVG(p,1) < 0 && VFlowAVG(p+1,1) < 0
             if VFlowAVG(p,1) < VFlowAVG(p+1,1)
                VFlowAVG(p,1)=VFlowAVG(p+1,1);
            else if VFlowAVG(p+1,1) < VFlowAVG(p,1)
                    VFlowAVG(p+1,1)=VFlowAVG(p,1);
                end
             end
            else
                if UFlowAVG(p,1) > UFlowAVG(p+1,1)
                    UFlowAVG(p,1)=UFlowAVG(p+1,1);
                else if UFlowAVG(p+1,1) > UFlowAVG(p,1)
                    UFlowAVG(p+1,1)=UFlowAVG(p,1);
                    end
                end
            end
        end
    end

    if UIndex3(p,1)==0 && UIndex2(p,1) ~= 0
        if UFlowAVG(p,1) > UFlowAVG(p-1,1)
            UFlowAVG(p,1)=UFlowAVG(p-1,1);
        else if UFlowAVG(p-1,1) > UFlowAVG(p,1)
                UFlowAVG(p-1,1)=UFlowAVG(p,1);
            end
        end
    end

    if VIndex3(p,1)==0 && VIndex2(p,1) ~= 0
        if VFlowAVG(p,1) > VFlowAVG(p-1,1)
            VFlowAVG(p,1)=VFlowAVG(p-1,1);
        else if VFlowAVG(p-1,1) > VFlowAVG(p,1)
                VFlowAVG(p-1,1)=VFlowAVG(p,1);
            end
        end
    end
end

%outlier motion not picked up
for p=2:patchnum
    if UFlowAVG(p,1)-UFlowAVG(p-1,1) >= 20
```

```
            UFlowAVG(p,1)=UFlowAVG(p-1,1);
        end
        if UFlowAVG(p,1)-UFlowAVG(p-1,1) <= -20
            UFlowAVG(p,1)=UFlowAVG(p-1,1);
        end
        if VFlowAVG(p,1)-VFlowAVG(p-1,1) >= 20
            VFlowAVG(p,1)=VFlowAVG(p-1,1);
        end
        if VFlowAVG(p,1)-VFlowAVG(p-1,1) <= -20
            VFlowAVG(p,1)=VFlowAVG(p-1,1);
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Part 6: Determine if flow measure passes constraints if not count
    %flags and determine FlowStatus 1=pass,0=fail
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if Flag == 1
        numflags=numflags+1;
        fprintf('FLAG\n');
        FlagIndex(count,1)=1;
    else
        Flag=0;
        numflags=0;
    end
    if numflags == 3
        FlowStatus=0;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Part7: Prediction or movement of edges with average optical flow from
each
    %patch if validation is passed and results fit smoothness criteria
    %inputs:
    %outputs:
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if FlowStatus == 1
        %move edges by average flow
        for k = 1:patchnum
            CurrEdges=cell2mat(PatchInfo(k,3));
            CurrStartCornerPts=cell2mat(PatchInfo(k,1));
            numedges=length(CurrEdges(:,1));
            for kk=1:numedges
                CurrEdges(kk,1)=CurrEdges(kk,1)+VFlowAVG(k,1);
                CurrEdges(kk,2)=CurrEdges(kk,2)+UFlowAVG(k,1);
            end
            PatchInfo(k,3)={CurrEdges};
            CurrStartCornerPts(1,1)=CurrStartCornerPts(1,1)+VFlowAVG(k,1);
            CurrStartCornerPts(1,2)=CurrStartCornerPts(1,2)+UFlowAVG(k,1);
            PatchInfo(k,1)={CurrStartCornerPts};
        end

        %move start and stop pts by flow average
        StartPt(1,1)=round(StartPt(1,1)+UFlowAVG(1,1));
        StartPt(2,1)=round(StartPt(2,1)+VFlowAVG(1,1));
        EndPt(1,1)=round(EndPt(1,1)+UFlowAVG(patchnum,1));
        EndPt(2,1)=round(EndPt(2,1)+VFlowAVG(patchnum,1));
```

```
        %re-draw grid and re-sample corner points
        diffY=abs(EndPt(2,1)-StartPt(2,1));
        diffX=abs(EndPt(1,1)-StartPt(1,1));
        sizeY=round(diffY/rowcount);
        sizeX=round(diffX/rowcount);
        if(mod(sizeX,2)>0)
            sizeX=sizeX+1;
        end
        if(mod(sizeY,2)>0)
            sizeY=sizeY+1;
        end

        %minimize amount of corners to within area near selected region
        ind5=0;
        for w = 1:length(Corners2Loc);
            if round(Corners2Loc(w,2)) >= (StartPt(2,1)-20) &&
round(Corners2Loc(w,2)) <= (EndPt(2,1)+20) ...
                    && round(Corners2Loc(w,1)) >= (StartPt(1,1)-20) &&
round(Corners2Loc(w,1)) <= (EndPt(1,1)+20)
                ind5=ind5+1;
                ReducedCorners(ind5,:)=[Corners2Loc(w,1) Corners2Loc(w,2) w];
            end
        end

        %re-sample corner points
        patchcount=0;
        for Ycount = 0:(rowcount-1)
            StartY=round(StartPt(2,1)+(sizeY*Ycount));
            for Xcount = 0:(rowcount-1)
                patchcount=patchcount+1;
                StartX=round(StartPt(1,1)+(sizeX*Xcount));
                StartCornerPt=[StartY,StartX];
                PatchInfo(patchcount,1)={StartCornerPt};

patchCorners=SaveCornerIndex(ReducedCorners,StartX,StartY,sizeX,sizeY);
                PatchInfo(patchcount,2)={patchCorners};
            end
        end

        %store flow values
        FlowAvgCorrected(count,1).U=UFlowAVG;
        FlowAvgCorrected(count,1).V=VFlowAVG;

    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Part8: Run SURF/RANSAC to get Homography to reset tracking process due
to
    %smoothness criteria failure
    %input:both image frames (I1/I2),thresholds for SURF and
RANSAC,FlowStatus
    %of fail (0)
    %output:patch edge index points moved by Homography (H) matrix
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if FlowStatus == 0

        %increment reset number
```

```
        numresets=numresets+1;

        %store flow as 0
        FlowAvgCorrected(count,1).U=0;
        FlowAvgCorrected(count,1).V=0;
        FlowAvg(count,1).U=0;
        FlowAvg(count,1).V=0;

        %use func to determine best frame to reset edge tracking pts
        FrameNum=DetermineBestFit2(ImgNIR,count);
        I1=histeq(ImgNIR(:,:,FrameNum));
        I2=histeq(ImgNIR(:,:,count));
        % Start SURF
            %set limit
            UpperLimit=550;
            LowerLimit=250;
            ind3=0;
            ind4=0;
            % Get the Key Points
              %Options.upright : Boolean which determines if we want a non-
rotation
              %invariant result (default false)
              Options.upright=true;
              %Options.upright=false;
              %Extended adds extra landmark position info to descriptor
              %Options.extended=true;
              %Options.tresh : Hessian response threshold (default 0.0002)-
changes the
              %#ofKeyPts
              Options.tresh=0.00035;%0.00055; %decreas=more pts,inc=less pts
              % Ipts : A struct w/ info about all detected key points, of 2
img's
              % Ipts.x , ipts.y : The landmark position
              Ipts1=OpenSurf(I1,Options);
              Ipts2=OpenSurf(I2,Options);
              %re sample Ipts1 and Ipts2 to include pts that are within
limits
              for aa=1:length(Ipts1)
                  Yvalue=round(Ipts1(aa).y);
                  if Yvalue >= LowerLimit && Yvalue <= UpperLimit
                      ind3=ind3+1;
                      RegionIpts1(ind3)=Ipts1(aa);
                  end
              end
              for aa=1:length(Ipts2)
                  Yvalue=round(Ipts2(aa).y);
                  if Yvalue >= LowerLimit && Yvalue <= UpperLimit
                      ind4=ind4+1;
                      RegionIpts2(ind4)=Ipts2(aa);
                  end
              end
            % Put the landmark descriptors in a matrix, index of
corresponding matching
            % pts-reshaped by taking Ipts.d(i) from i->length(Ipts) and
storing into D
            % as D1=64xlength(Ipts1) and D2=64xlength(Ipts2)
              D1 = reshape([RegionIpts1.descriptor],64,[]);
```

```
              D2 = reshape([RegionIpts2.descriptor],64,[]);
            % Find the best matches-step through length of Ipts1 or key pts
found
              err=zeros(1,length(RegionIpts1));
              cor1=1:length(RegionIpts1); %index of all Ipts1 values
              cor2=zeros(1,length(RegionIpts1));
              %re-orders D1 to length of D2, then calc distance from D2 to
D1,
              %then sums every pt with 1, stores minimum distance value as
err(i) and
              %cor2 as index of that value,if err(i)<0.05 get rid of those
matches
              for i=1:length(RegionIpts1),
                  distance=sum((D2-repmat(D1(:,i),[1
length(RegionIpts2)])).^2,1);
                  [err(i),cor2(i)]=min(distance);
                  if err(i)<0.05
                    D2(:,cor2(i))=1000;
                  end
              end
            % Sort matches on vector distance
            % Sort err in ascending order and stores index as ind, to sort
cor1 and
            % cor2 in the same ascending order
              [err, ind]=sort(err);
              cor1=cor1(ind);
              cor2=cor2(ind);
        %     % Show both images, create blank image as rows, column*2, color
as I1
%               I = zeros([size(I1,1) size(I1,2)*2 size(I1,3)]);
%               I(:,1:size(I1,2),:)=I1;
I(:,size(I1,2)+1:size(I1,2)+size(I2,2),:)=I2;
%               figure, imshow(I/255); hold on;
%               title('SURF Feature Matches Plot');
%               plot(x,y,'g-','MarkerSize',15);
%         %     % Show the best matches(i<=cor1/2) + save pts from both image
frames
%         %     % the best matches show up as the 2 img slices side-by-side
%         %     % Ipts1(cor1(i)).x=img1 matching pt x value,y=img1 matching
pt y value
%         %     % Ipts2(cor2(i)).x=img2 matching pt x
value+(length(img1)),y=value of img2
%               for i=1:200,
%                   c=rand(1,3);
%                   plot([RegionIpts1(cor1(i)).x
RegionIpts2(cor2(i)).x+size(I1,2)],[RegionIpts1(cor1(i)).y
RegionIpts2(cor2(i)).y],'-','Color',c)
%                   plot([RegionIpts1(cor1(i)).x
RegionIpts2(cor2(i)).x+size(I1,2)],[RegionIpts1(cor1(i)).y
RegionIpts2(cor2(i)).y],'o','Color',c)
%               end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %Part1:DONE

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %PART2:
            %input:SURF Features as two vectors, Ipts1 and Ipts2,ROI vector
            %ouput:RANSAC results for inlier detection/outlier rejection,
display
            %results           = structure containing the following fields:
            %
            %   Theta               = estimated parameter vector
            %   E                   = fitting error obtained from man_fun
            %   CS                  = consensus set (true -> inliers, false -
> outliers)
            %   r                   = rank of the solution
            %   iter                = number of iterations
            %   time                = time to perform the computation
            %summary:RANSAC

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %init
            X1=zeros(1,length(RegionIpts1));
            Y1=zeros(1,length(RegionIpts1));
            X2=zeros(1,length(RegionIpts1));
            Y2=zeros(1,length(RegionIpts1));

            %use length of Ipts2, bc ransac input data must have same length,
Ipts2 is smaller
            for j=1:length(RegionIpts1)
                X1(j)=RegionIpts1(cor1(j)).x;
                Y1(j)=RegionIpts1(cor1(j)).y;
            end
            for k=1:length(RegionIpts1)
                X2(k)=RegionIpts2(cor2(k)).x;
                Y2(k)=RegionIpts2(cor2(k)).y;
            end

            %input data for RANSAC
            X1bar=[X1;Y1];
            X2bar=[X2;Y2];

            %Format input data vector as 4 rows [X1;Y1:X2;Y2]
            TestData=[X1bar;X2bar];

            %Implement RANSAC
            % set RANSAC options
            options.epsilon = 1e-6;
            %threshold (default 1-1e-5), lower=tighter thresh, less pts,
higher=loose
            %threshold or more pts as inliers included
            options.P_inlier = 1-1e-5;
            options.sigma = 1;
            options.validateMSS_fun = @validateMSS_homography;
            options.est_fun = @estimate_homography;
            options.man_fun = @error_homography;
            options.mode = 'MSAC';
            options.Ps = [];
            options.notify_iters = [];
            options.min_iters = 1000;
```

```
            options.fix_seed = false;
            options.reestimate = true;
            options.stabilize = false;

            % run RANSAC
            [results, options] = RANSACzz(TestData, options);

            %move start/stop
            SaveStart=StartPt;
            SaveEnd=EndPt;
            [StartPt(1,1),StartPt(2,1)] = mapping_homography(SaveStart(1,1),
SaveStart(2,1), true, results.Theta);
            [EndPt(1,1),EndPt(2,1)] = mapping_homography(SaveEnd(1,1),
SaveEnd(2,1), true, results.Theta);

            %keep homography estimate bounded
            if StartPt(1,1)-SaveStart(1,1) >= 5 || StartPt(1,1)-
SaveStart(1,1) <= -20
                StartPt(1,1)=SaveStart(1,1);
            end

            if EndPt(1,1)-SaveEnd(1,1) >= 20 || EndPt(1,1)-SaveEnd(1,1) <= -8
                EndPt(1,1)=SaveEnd(1,1);
            end

            if StartPt(2,1)-SaveStart(2,1) >= 5 || StartPt(2,1)-
SaveStart(2,1) <= -8
                StartPt(2,1)=SaveStart(2,1);
            end

            if EndPt(2,1)-SaveEnd(2,1) >= 8 || EndPt(2,1)-SaveEnd(2,1) <= -5
                EndPt(2,1)=SaveEnd(2,1);
            end

            if StartPt(1,1) <= 145
                StartPt(1,1)=StartPt(1,1)+10;
            end

            if StartPt(2,1) <= 300
                StartPt(2,1)=StartPt(2,1)+10;
            end

            if EndPt(2,1) >= 500
                EndPt(2,1)=EndPt(2,1)-10;
            end

            if EndPt(1,1) >= 520
                EndPt(1,1)=EndPt(1,1)-10;
            end

            %re-draw grid and re-sample corner points
            diffY=abs(EndPt(2,1)-StartPt(2,1));
            diffX=abs(EndPt(1,1)-StartPt(1,1));
            sizeY=round(diffY/rowcount);
            sizeX=round(diffX/rowcount);
            if(mod(sizeX,2)>0)
                sizeX=sizeX+1;
```

```
            end
            if(mod(sizeY,2)>0)
                sizeY=sizeY+1;
            end

            %re-sample corner pts into patches that have moved
            %minimize amount of corners to within area near selected region
            ind5=0;
            for w = 1:length(Corners2Loc);
                if round(Corners2Loc(w,2)) >= (StartPt(2,1)-20) &&
round(Corners2Loc(w,2)) <= (EndPt(2,1)+20) ...
                        && round(Corners2Loc(w,1)) >= (StartPt(1,1)-20) &&
round(Corners2Loc(w,1)) <= (EndPt(1,1)+20)
                    ind5=ind5+1;
                    ReducedCorners(ind5,:)=[Corners2Loc(w,1) Corners2Loc(w,2)
w];
                end
            end

            %apply blur + edge detection on region, re-acquire
            %corners + edges
            I2Blur=imgaussfilt(I2,0.5);
            I2BlurEdges=edge(I2Blur,'canny',[0.035 0.0675]);

            patchcount=0;
            for Ycount = 0:(rowcount-1)
                StartY=round(StartPt(2,1)+(sizeY*Ycount));
                for Xcount = 0:(rowcount-1)
                    patchcount=patchcount+1;
                    index2=0;
                    StartX=round(StartPt(1,1)+(sizeX*Xcount));
                    StartCornerPt=[StartY,StartX];
                    PatchInfo(patchcount,1)={StartCornerPt};

patchCorners=SaveCornerIndex(ReducedCorners,StartX,StartY,sizeX,sizeY);
                    PatchInfo(patchcount,2)={patchCorners};
                    patchOutline=I2BlurEdges(StartY:((StartY+sizeY)-
1),StartX:((StartX+sizeX)-1));
                    patchEdges=SaveEdgeIndex(patchOutline,StartX,StartY);
                    EdgeNum(patchcount,1)=length(patchEdges(:,1));
                    PatchInfo(patchcount,3)={patchEdges};
                end
            end

            Flag=0;
            numflags=0;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Part9: Highlight edges on next frame using 'HighlightEdges.m'
    %function and write image to file
    %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %display edges on second image, use round final index
    %display results on second image, use round final index
    Segout3=I2;
    for k = 1:patchnum
            CurrEdges=cell2mat(PatchInfo(k,3));
```

```
            CurrCorners=cell2mat(PatchInfo(k,2));
            numedges=length(CurrEdges(:,1));
             if numedges >= 10
                    for kk=1:numedges
                        Yind=round(CurrEdges(kk,1));
                        Xind=round(CurrEdges(kk,2));
                        if Yind > (StartPt(2,1)-50) && Yind < (EndPt(2,1)+50)
&& Xind > 0 && Xind < 800
                            Segout3(Yind,Xind)=0;
                        end
                    end
             end
    end
    Flag=0;

%     figure,imshow(I2),hold on;
%     %make patch grid
%     for k = StartPt(2,1):sizeY:(EndPt(2,1)+2)
%         xgrid = [StartPt(1,1) (EndPt(1,1)+2)];
%         ygrid = [k k];
%         plot(xgrid,ygrid,'Color','g','LineStyle','-');
%         plot(xgrid,ygrid,'Color','r','LineStyle',':');
%     end
%
%     for k = StartPt(1,1):sizeX:(EndPt(1,1)+2)
%         xgrid = [k k];
%         ygrid = [StartPt(2,1) (EndPt(2,1)+2)];
%         plot(xgrid,ygrid,'Color','g','LineStyle','-');
%         plot(xgrid,ygrid,'Color','r','LineStyle',':');
%     end
%
%     for mm=1:patchnum
%         Corners=cell2mat(PatchInfo(mm,2));
%         for m = 1:length(Corners(:,1))
%             if Corners(1,3)~=0
%                 PlotCorners=round(Corners(m,:));
%                 plot(PlotCorners(:,1),PlotCorners(:,2),'.m');
%             end
%         end
%     end
%     title('Next Frame Grid with Corners');
%
%     fprintf('Frame %d \n',count);
%     figure,imshow(Segout3),hold on;
%     string2=['Frame' num2str(count) 'with Edges Highlighted'];
%     title(string2);

    %write Segout2(jpeg) image with edges highlighted to directory specified
    %name current segment
    string=['C:\ECU\Thesis\DataResultsSet5\TimedTest1\Frame' num2str(count)
'.jpeg'];
    imwrite(Segout3,string);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     %add menu to step to next frame
%     choice=menu('Continue? Press Yes or No','Yes','No');
%     if choice==2 || choice==0
```

```
%         break;
%     end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
toc;
fprintf('\n All Image Frames Complete! \n');
EndingTime=datestr(now)
```

Determine Patch Number Function:

```
function
[FinalPatchNum,MaxUHorizThresh,MaxVHorizThresh,MaxUVertThresh,MaxVVertThresh,
NumCorners] =
DeterminePatchNum4(img1,ImgCycle,StartPt,EndPt,ReducedCorners,corners,MinPatc
hCorners)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PatchCorners = SaveCornerIndex(corners,startX,startY)
%inputs:
%Reduced Corners = struct with corners across patch area + surrounding [x y
%index]
%startPt and EndPt = starting points for the selected region
%img1 and ImgCycle = first frame and the rest of frames to make up 1 cycle
%outputs:
%patch number to satisfy constraints, U/V velocity thresholds for each
%patch
%summary:
%this function computes the correct patch number to track a ROI and
%determines a threshold for continuous motion between neighbors
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initialize tracker
tracker3=vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker3,corners.Location,img1);
CornersStruct=struct;
num=2;
CycleNum=size(ImgCycle,3);
[Corners2Loc,validity]=step(tracker3,ImgCycle(:,:,num));
UDiffStruct=struct;
VDiffStruct=struct;
%begin loop to determine patchnumber that has all patches pass chi-square
%GOF test to be normally distributed
NormalStatus = 0;
MinCornerStatus=0;
i=3;
tic;
while MinCornerStatus == 0
    while NormalStatus == 0
        if i < 13
            i=i+1;
        end
        patchnum=i^2;
        rowcount=sqrt(patchnum);
        %set patch size
        diffY=abs(EndPt(2,1)-StartPt(2,1));
        diffX=abs(EndPt(1,1)-StartPt(1,1));
        sizeY=round(diffY/rowcount);
        sizeX=round(diffX/rowcount);
```

```
        if(mod(sizeX,2)>0)
            sizeX=sizeX+1;
        end
        if(mod(sizeY,2)>0)
            sizeY=sizeY+1;
        end

        patchcount=0;
        for Ycount = 0:(rowcount-1)
            StartY=round(StartPt(2,1)+(sizeY*Ycount));
            for Xcount = 0:(rowcount-1)
                patchcount=patchcount+1;
                StartX=round(StartPt(1,1)+(sizeX*Xcount));

patchCorners=SaveCornerIndex(ReducedCorners,StartX,StartY,sizeX,sizeY);
                CornersStruct(patchcount).CornerIndex=patchCorners;
            end
        end

        NumCorners=zeros(patchnum,1);
        for p = 1:patchnum
            CurrCorners=CornersStruct(p).CornerIndex;
            number=length(CurrCorners(:,1));
            NumCorners(p,1)=number;
        end

        %save flow in each patch + calculate average flow in X and Y
        VFlowAVG = zeros(patchnum,1);
        UFlowAVG = zeros(patchnum,1);
        CornerN=zeros(patchnum,1);
        PatchCornerFlowU=struct;
        PatchCornerFlowV=struct;
        for k = 1:patchnum
            CurrCorners=CornersStruct(k).CornerIndex;
            Test=CurrCorners(1,1);
            if Test == 0
                PatchCornerFlowU(k).U=0;
                PatchCornerFlowV(k).V=0;
                UFlowAVG(k,:)=0;
                VFlowAVG(k,:)=0;
            else
                numcorners=length(CurrCorners(:,1));
                CornerN(k,1)=numcorners;
                PatchU=zeros(numcorners,1);
                PatchV=zeros(numcorners,1);
                for kk=1:numcorners
                    MovedIndex=CurrCorners(kk,3);
                    PatchU(kk)=Corners2Loc(MovedIndex,1)-CurrCorners(kk,1);
                    PatchV(kk)=Corners2Loc(MovedIndex,2)-CurrCorners(kk,2);
                end
                UFlowAVG(k,:)=mean(PatchU);
                VFlowAVG(k,:)=mean(PatchV);
                PatchCornerFlowU(k).U=PatchU;
                PatchCornerFlowV(k).V=PatchV;
            end
        end
```

```
%save positive or negative flow values and save neighbor flow if flow
%in X or Y is = 0
for k=1:patchnum
    U=PatchCornerFlowU(k).U;
    V=PatchCornerFlowV(k).V;
        for kk = 1:length(U)
            if U(kk)==0
                ind1=kk;
                if ind1 > 1
                    ind1=ind1-1;
                else if ind1 == 1
                    for w=2:length(U)
                        if U(w) ~= 0
                            ind1=w;
                        end
                    end
                else
                    ind1=length(U);
                    end
                end
                U(kk)=U(ind1);
            end
        end
        for kk = 1:length(V)
            if V(kk)==0
                ind2=kk;
                if ind2 > 1
                    ind2=ind2-1;
                else if ind2 == 1
                    for w=2:length(V)
                        if V(w) ~= 0
                            ind2=w;
                        end
                    end
                else
                    ind2=length(V);
                    end
                end
                V(kk)=V(ind2);
            end
        end
        PatchCornerFlowU(k).U=U;
        PatchCornerFlowV(k).V=V;
        %calculate new avg flow in x and y
        UFlowAVG(k,:)=mean(PatchCornerFlowU(k).U);
        VFlowAVG(k,:)=mean(PatchCornerFlowV(k).V);
end

%run chi goodness of fit test to check if each flow is normal dist.
UFlowChiTest=zeros(patchnum,1);
VFlowChiTest=zeros(patchnum,1);
for pp=1:patchnum
    U=PatchCornerFlowU(pp).U;
    UFlowChiTest(pp,1)=chi2gof(U);
    V=PatchCornerFlowV(pp).V;
    VFlowChiTest(pp,1)=chi2gof(V);
end
```

```
        ind9=0;
        ind10=0;
        for q=1:length(UFlowChiTest(:,1))
            if UFlowChiTest(q) == 0
                ind9=ind9+1;
            end
            if VFlowChiTest(q) == 0
                ind10=ind10+1;
            end
        end

        if ind9 == length(UFlowChiTest(:,1)) && ind10 ==
length(VFlowChiTest(:,1))
            NormalStatus=1;
            FinalPatchNum=patchnum;
        end
    end
    if min(NumCorners(:,1)) >= MinPatchCorners
        MinCornerStatus=1;
    else
        StartPt(1,1)=StartPt(1,1)-2;
        StartPt(2,1)=StartPt(2,1)-2;
        EndPt(1,1)=EndPt(1,1)+2;
        EndPt(2,1)=EndPt(1,1)+2;
        NormalStatus=0;
        i=i-1;
    end
end
toc;
fprintf('Patch Count Determined!: %d\n',patchnum);

%begin loop to step through frame 2-end of ImgCycle and calculate the U and V
%difference between horizontal and verical neighbors. store all vel
%differences into a struct or cell array to make up zero mean Gaussian
HorizDiffU=zeros(rowcount*(rowcount-1),CycleNum-1);
VertDiffU=zeros(rowcount*(rowcount-1),CycleNum-1);
HorizDiffV=zeros(rowcount*(rowcount-1),CycleNum-1);
VertDiffV=zeros(rowcount*(rowcount-1),CycleNum-1);
num=1;
tracker2=vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker2,corners.Location,img1);
Frameind=0;
while num < CycleNum
    num=num+1;
    Frameind=Frameind+1;
    [NewCornersLoc,validity]=step(tracker2,ImgCycle(:,:,num));
    UDiff=zeros(((rowcount-1)*rowcount)*2,1);
    VDiff=zeros(((rowcount-1)*rowcount)*2,1);
    ind7=0;
    ind8=0;
    ind11=0;
    ind12=0;
    ind13=0;
    ind14=0;
    %save flow in each patch + calculate average flow in X and Y
    VFlowAVG = zeros(patchnum,1);
    UFlowAVG = zeros(patchnum,1);
```

```
CornerN=zeros(patchnum,1);
PatchCornerFlowU=struct;
PatchCornerFlowV=struct;
for k = 1:patchnum
    CurrCorners=CornersStruct(k).CornerIndex;
    Test=CurrCorners(1,1);
    if Test == 0
        PatchCornerFlowU(k).U=0;
        PatchCornerFlowV(k).V=0;
        UFlowAVG(k,:)=0;
        VFlowAVG(k,:)=0;
    else
        numcorners=length(CurrCorners(:,1));
        CornerN(k,1)=numcorners;
        PatchU=zeros(numcorners,1);
        PatchV=zeros(numcorners,1);
        for kk=1:numcorners
            MovedIndex=CurrCorners(kk,3);
            PatchU(kk)=NewCornersLoc(MovedIndex,1)-CurrCorners(kk,1);
            PatchV(kk)=NewCornersLoc(MovedIndex,2)-CurrCorners(kk,2);
        end
        UFlowAVG(k,:)=mean(PatchU);
        VFlowAVG(k,:)=mean(PatchV);
        PatchCornerFlowU(k).U=PatchU;
        PatchCornerFlowV(k).V=PatchV;
    end
end
%save positive or negative flow values and save neighbor flow if flow
%in X or Y is = 0
for k=1:patchnum
    U=PatchCornerFlowU(k).U;
    V=PatchCornerFlowV(k).V;
        for kk = 1:length(U)
            if U(kk)==0
                ind1=kk;
                if ind1 > 1
                    ind1=ind1-1;
                else if ind1 == 1
                    for w=2:length(U)
                        if U(w) ~= 0
                            ind1=w;
                        end
                    end
                else
                    ind1=length(U);
                    end
                end
                U(kk)=U(ind1);
            end
        end
        for kk = 1:length(V)
            if V(kk)==0
                ind2=kk;
                if ind2 > 1
                    ind2=ind2-1;
                else if ind2 == 1
                    for w=2:length(V)
```

```
                        if V(w) ~= 0
                            ind2=w;
                        end
                    end
                else
                    ind2=length(V);
                    end
                end
                V(kk)=V(ind2);
            end
        end
        PatchCornerFlowU(k).U=U;
        PatchCornerFlowV(k).V=V;
        %calculate new avg flow in x and y
        UFlowAVG(k,:)=mean(PatchCornerFlowU(k).U);
        VFlowAVG(k,:)=mean(PatchCornerFlowV(k).V);
end
%calculate U and V difference between neighbors and store in struct or
%cell array?
%compare both U and V directions
UAVGDiff=zeros(rowcount,rowcount-1);
VAVGDiff=zeros(rowcount,rowcount-1);
for k=1:rowcount
    ind6=0;
    if k == 1
        CurrRow=1;
    else
        CurrRow=(k*rowcount)-(rowcount-1);
    end
    for kk=CurrRow:(CurrRow+(rowcount-2))
        ind6=ind6+1;
        ind7=ind7+1;
        ind11=ind11+1;
        UPatchDiff=UFlowAVG(kk,1)-UFlowAVG(kk+1,1);
        UAVGDiff(k,ind6)=UPatchDiff;
        %store difference
        UDiff(ind7,1)=UPatchDiff;
        HorizDiffU(ind11,Frameind)=UPatchDiff;
    end
end

for k=1:rowcount
    ind6=0;
    if k == 1
        CurrRow=1;
    else
        CurrRow=(k*rowcount)-(rowcount-1);
    end
    for kk=CurrRow:(CurrRow+(rowcount-2))
        ind6=ind6+1;
        ind8=ind8+1;
        ind12=ind12+1;
        VPatchDiff=VFlowAVG(kk,1)-VFlowAVG(kk+1,1);
        VAVGDiff(k,ind6)=VPatchDiff;
        %store difference
        VDiff(ind8,1)=VPatchDiff;
        HorizDiffV(ind12,Frameind)=VPatchDiff;
```

```
    end
end

%check variability across patches vertically
%compare both U and V directions
UAVGDiff2=zeros(rowcount-1,rowcount);
VAVGDiff2=zeros(rowcount-1,rowcount);
for k=1:rowcount
    ind6=0;
    CurrCol=k;
    for kk=CurrCol:rowcount:((rowcount-2)*rowcount)+CurrCol
        ind6=ind6+1;
        ind7=ind7+1;
        UPatchDiff2=UFlowAVG(kk,1)-UFlowAVG(kk+rowcount,1);
        UAVGDiff2(ind6,k)=UPatchDiff2;
        %store difference in struct
        UDiff(ind7,1)=UPatchDiff2;
    end
end

for k=1:length(UAVGDiff2(:,1))
    for kk = 1:length(UAVGDiff2(1,:))
        ind13=ind13+1;
        VertDiffU(ind13,Frameind)=UAVGDiff2(k,kk);
    end
end

for k=1:rowcount
    ind6=0;
    CurrCol=k;
    for kk=CurrCol:rowcount:((rowcount-2)*rowcount)+CurrCol
        ind6=ind6+1;
        ind8=ind8+1;
        VPatchDiff2=VFlowAVG(kk,1)-VFlowAVG(kk+rowcount,1);
        VAVGDiff2(ind6,k)=VPatchDiff2;
        %save difference in struct
        VDiff(ind8,1)=VPatchDiff2;
    end
end

for k=1:length(VAVGDiff2(:,1))
    for kk = 1:length(VAVGDiff2(1,:))
        ind14=ind14+1;
        VertDiffV(ind14,Frameind)=VAVGDiff2(k,kk);
    end
end

UDiffStruct(Frameind).U=UDiff;
VDiffStruct(Frameind).V=VDiff;

    %move start and stop pts by flow average
    StartPt(1,1)=round(StartPt(1,1)+UFlowAVG(1,1));
    StartPt(2,1)=round(StartPt(2,1)+VFlowAVG(1,1));
    EndPt(1,1)=round(EndPt(1,1)+UFlowAVG(patchnum,1));
    EndPt(2,1)=round(EndPt(2,1)+VFlowAVG(patchnum,1));

    %re-draw grid and re-sample corner points
```

```
        diffY=abs(EndPt(2,1)-StartPt(2,1));
        diffX=abs(EndPt(1,1)-StartPt(1,1));
        sizeY=round(diffY/rowcount);
        sizeX=round(diffX/rowcount);
        if(mod(sizeX,2)>0)
            sizeX=sizeX+1;
        end
        if(mod(sizeY,2)>0)
            sizeY=sizeY+1;
        end

        %minimize amount of corners to within area near selected region
        ind5=0;
        for w = 1:length(NewCornersLoc);
            if round(NewCornersLoc(w,2)) >= (StartPt(2,1)-20) &&
round(NewCornersLoc(w,2)) <= (EndPt(2,1)+20) ...
                    && round(NewCornersLoc(w,1)) >= (StartPt(1,1)-20) &&
round(NewCornersLoc(w,1)) <= (EndPt(1,1)+20)
                ind5=ind5+1;
                ReducedCorners2(ind5,:)=[NewCornersLoc(w,1)
NewCornersLoc(w,2) w];
            end
        end

        %re-sample corner pts
        patchcount=0;
        for Ycount = 0:(rowcount-1)
            StartY=round(StartPt(2,1)+(sizeY*Ycount));
            for Xcount = 0:(rowcount-1)
                patchcount=patchcount+1;
                StartX=round(StartPt(1,1)+(sizeX*Xcount));

patchCorners=SaveCornerIndex(ReducedCorners2,StartX,StartY,sizeX,sizeY);
                CornersStruct(patchcount).CornerIndex=patchCorners;
            end
        end
end
toc;
fprintf('One Cycle Finished!\n');

%determine distribution of all patch differences
HorizDiffU2=zeros(patchnum,Frameind);
HorizDiffV2=zeros(patchnum,Frameind);
ind6=0;
for k=1:rowcount
        if k == 1
            CurrRow=1;
        else
            CurrRow=(k*rowcount)-(rowcount-1);
        end
        for kk=CurrRow:(CurrRow+(rowcount-2))
            ind6=ind6+1;
            HorizDiffU2(kk,:)=HorizDiffU(ind6,:);
            HorizDiffV2(kk,:)=HorizDiffV(ind6,:);
        end
end
VertDiffU2=zeros(patchnum,Frameind);
```

```
VertDiffV2=zeros(patchnum,Frameind);
for p=1:(patchnum-rowcount)
    VertDiffU2(p,:)=VertDiffU(p,:);
    VertDiffV2(p,:)=VertDiffV(p,:);
end

% %run chi-goodness of fit test on each patch difference in U and V
UNormalDistTestH=zeros(patchnum,1);
UNormalDistTestV=zeros(patchnum,1);
VNormalDistTestH=zeros(patchnum,1);
VNormalDistTestV=zeros(patchnum,1);
UHorizThresh=zeros(patchnum,1);
UVertThresh=zeros(patchnum,1);
VHorizThresh=zeros(patchnum,1);
VVertThresh=zeros(patchnum,1);
for aa=1:patchnum
    if mod(aa,rowcount)~=0
      UNormalDistTestH(aa,1)=chi2gof(HorizDiffU2(aa,:));
      VNormalDistTestH(aa,1)=chi2gof(HorizDiffV2(aa,:));
      if CornerN(aa,1) <= 8
        UHorizThresh(aa,1)=0;
        VHorizThresh(aa,1)=0;
      else

UHorizThresh(aa,1)=(mean(HorizDiffU2(aa,:))+(2*std(HorizDiffU2(aa,:))));

VHorizThresh(aa,1)=(mean(HorizDiffV2(aa,:))+(2*std(HorizDiffV2(aa,:))));
      end
    end
end
for aa=1:(patchnum-rowcount)
      UNormalDistTestV(aa,1)=chi2gof(VertDiffU2(aa,:));
      VNormalDistTestV(aa,1)=chi2gof(VertDiffV2(aa,:));
      if CornerN(aa,1) <= 8
        UHorizThresh(aa,1)=0;
        VHorizThresh(aa,1)=0;
      else
        UVertThresh(aa,1)=(mean(VertDiffU2(aa,:))+(2*std(VertDiffU2(aa,:))));
        VVertThresh(aa,1)=(mean(VertDiffV2(aa,:))+(2*std(VertDiffV2(aa,:))));
      end
end

MaxUHorizThresh=max(UHorizThresh);
MaxVHorizThresh=max(VHorizThresh);
MaxUVertThresh=max(UVertThresh);
MaxVVertThresh=max(VVertThresh);

end
```

Determine Best Fit Function:

```
function FrameNum = DetermineBestFit2(ImgNIR,count)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FrameNum = DetermineBestFit(I2,I1,BestFrame)%
```

```
%inputs:
%ImgNIR=image frame data set
%count=current frame count
%
%outputs:
% FrameNum=frame with highest matching SURF features
%
%summary:
%this function determines the best frame to match the current frame with
%the most SURF features
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PrevCt=0;
PtCt=1;
FrameCt=0;
fprintf('Determine Best Fit\n');
while(PrevCt < PtCt)
    FrameCt=FrameCt+1;
    Prev=count-FrameCt;
    if Prev > 0
        I1=histeq(ImgNIR(:,:,Prev));
        I2=histeq(ImgNIR(:,:,count));
        PrevCt=PtCt;
        % Start SURF
        % Get the Key Points
        %Options.upright : Boolean which determines if we want a non-rotation
        %invariant result (default false)
        Options.upright=true;
        %Options.upright=false;
        %Extended adds extra landmark position info to descriptor
        %Options.extended=true;
        %Options.tresh : Hessian response threshold (default 0.0002)-changes
the
        %#ofKeyPts
        Options.tresh=0.00035;%0.00055; %decreas=more pts,inc=less pts
        % Ipts : A struct w/ info about all detected key points, of 2 img's
        % Ipts.x , ipts.y : The landmark position
        Ipts1=OpenSurf(I1,Options);
        Ipts2=OpenSurf(I2,Options);
        % Put the landmark descriptors in a matrix, index of corresponding
matching
        % pts-reshaped by taking Ipts.d(i) from i->length(Ipts) and storing
into D
        % as D1=64xlength(Ipts1) and D2=64xlength(Ipts2)
          D1 = reshape([Ipts1.descriptor],64,[]);
          D2 = reshape([Ipts2.descriptor],64,[]);
        % Find the best matches-step through length of Ipts1 or key pts found
          err=zeros(1,length(Ipts1));
          cor1=1:length(Ipts1); %index of all Ipts1 values
          cor2=zeros(1,length(Ipts1));
          %re-orders D1 to length of D2, then calc distance from D2 to D1,
          %then sums every pt with 1, stores minimum distance value as err(i)
and
          %cor2 as index of that value,if err(i)<0.05 get rid of those
matches
          for i=1:length(Ipts1),
              distance=sum((D2-repmat(D1(:,i),[1 length(Ipts2)])).^2,1);
              [err(i),cor2(i)]=min(distance);
```

```
            if err(i)<0.05
              D2(:,cor2(i))=1000;
            end
          end
        % Sort matches on vector distance
        % Sort err in ascending order and stores index as ind, to sort cor1
and
        % cor2 in the same ascending order
        [err, ind]=sort(err);
        cor1=cor1(ind);
        cor2=cor2(ind);
        PtCt=length(cor1);
    end
end
FrameNum=Prev+1;
% FrameNum=FrameCt-1;
end
```