Department of Computer Science

Retinopathy

Kwan Ho Ho

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering

September 2019 | CSMSC-19

Declaration:

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Kwan Ho Ho, September 2019

EXECUTIVE SUMMARY:

Diabetic Retinopathy (DR) is one of the eye-related disease that reduces the integrity of the blood vessels in the retinal layers which leads to retinal blood vessel leakage [2]. Sodium Fluorescein Angiography (**FA**) is widely used to monitor the leakage or the permeability of the vessel by imaging the back of the eyes as an important diagnostic value. Gamez [2] which is a PhD student in University of Bristol started FA on mice. Gamez [2] manually extracted fluorescent intensity data from the resulting FA videos and a graph of the fluorescence intensity ratio (**FIR**) versus time was plotted to obtain the gradient which is the solute flux ($\Delta I_f / \Delta t$). These data was then used to assist the development of the Fick's Law adapted equation $P=\Delta I_f / \Delta t / (\Delta C \times A)$ to obtain the permeability of the vessel. The obstacle of this method was the manual data capturing process was too time consuming. This method also requires a lot of manual adjustments due to the movement of the camera caused by the heartbeat of the mice and their eyeball motion. The movement of the camera also caused blurry and unsharp images in the FA videos which led to inaccurate fluorescent intensity. A more intelligent way of data capturing was developed in this project using openCV with Python.

This project firstly experimented on using K-means clustering to segment the exchange vessel groups and the large vessel group out of the FA frames to obtain the FIR for the FIR vs time graph. This project experimented on the two settings of K-mean clustering. One used random initial centers and the other one used the previous frame's centers found by K-means clustering as the initial centers of the K-means clustering for the current frame (Reuse center K-means clustering). The experiment found that the random initial centers K-means clustering output stable FIR when the maximum iteration was 7 or above and the best epsilon (specific accuracy) was 0.1. Maximum iteration below 7 cannot be used due to FIR vs time graph showed large amount of noise and severe deformation. Conversely, the reuse center K-means clustering showed no deformation and noise on the FIR vs time graph when the maximum iteration was 7 or below and a much shorter execution time than the random initial centers K-means clustering. Then, the difference on the gradient of the FIR vs time graph was further examine between the two K-means clustering setting. The random initial centers K-means clustering showed fluctuation on the gradient value when the maximum iteration was between 7 and 15. The reuse center K-means clustering showed either an ascending or descending trend on the gradient value when the maximum iteration was below 7 and the gradient value stabilized when the maximum iteration was between 7 and 15. Reuse center K-means clustering was decided to implement in the final software and maximum iteration 7 was set as default to prioritize gradient accuracy over the execution time, and allow user to lower the maximum iteration to reduce execution time. This project then experimented on blurry frame classification by using Sobel edge detection. Convolution was performed with Sobel derivative operator on each FA frame to obtain an edge sharpness value. The edge sharpness versus frame number graphs were examined for all video and discovered a great separation between sharp and blurry frames in edge sharpness value. Sharp frames had higher edge sharpness and blurry frames had lower edge sharpness. A piece of code was created to loop along all the data points in edge sharpness vs frame number graph to classify sharp and blurry frames. The code firstly checked if the range of several neighboring data point (PtPbox) is larger than a specific value (tolerance value), then the data point needed a sharpness check, which take the mean of several neighboring data point (meanBox) and check is the current data point edge sharpness is lower or higher than the mean value. Lower means blurry frame and higher means sharp frames. A series of experiments were performed and the optimal value for PtPbox is 20, the meanBox is 10, the tolerance value is 0.1 and no histogram equalization is required. The sharp frames identification accuracy was above 80% and the blurry frames identification accuracy was above 96% for all the tested FA videos.

All these experimental codes were then connected by a graphical user interface based on python with PyQT4. Finally, the PyInstaller was used to package these Python codes into a stand-alone Microsoft Window executable for Gamez [2] to use.

Index Page:

Chapter 1: Introduction and Project Aim

1.1 Introduction:

Diabetic Retinopathy (DR) is one of the eye-related disease that occurs during diabetes. DR affects the retina of the eye, a light sensitive layer found at the back of the eye which helps to produce visual output through the optic nerve [1]. DR reduces the integrity of the blood vessels in the retinal layers which leads to hemorrhages, abnormal growth of blood vessels, blood vessel leakage and hard exudates, among other microvascular complications [2]. These complications can eventually lead to permanent blindness. Therefore, an early detection of DR is crucial for patients with diabetes to prevent deterioration of their eyesight. One of the most effective ways to observe DR is through monitoring the leakage or the permeability of the retinal vessel by imaging the back of the eyes [3]. Sodium fluorescein angiography (FA) is one of the most effective imaging techniques to be used. Sodium fluorescein is administered intravenously to the diabetic patient, following by an observation on the circulation of the dye thorough recording the retina capillaries [4]. Healthy retinas should show no dye leakage. Retinas from patients with DR will demonstrate certain amount of dye leakage due to the breakdown in the blood-retina barrier [4].

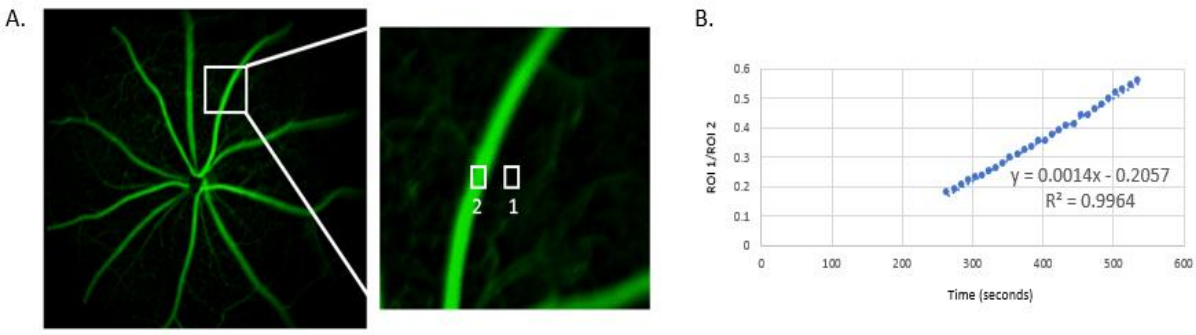


Figure 1: NaF Angiography Analysis [2]. A. A chosen area of large vessel in retina for analysis. Region of interest in large vessel (box 2) is measured over time to normalize for fluorescent intensity during analysis. Interstitium (exchange vessels) (box 1) are kept consist throughout course of measurement. B. The ratio of mean fluorescence intensity in box 1 over box 2 is plotted to determine slope of the line, i.e. solute flux which is used for the calculation of apparent permeability

Gamez [2] which is a PhD student in University of Bristol started Optical Coherence Tomography (OCT) and FA on mice. Firstly, the mice were anesthetized and given Sodium Fluorescein (NaF). After the injection of the dye, the vasculature of the retina started to fill up until it was saturated. The process was recorded right after NaF injection. Then the videos were processed to extract information, two boxes of certain area of the vessel was selected (Figure 1A). One box was the region of interest in the large vessel (box 2) and the other one was the exchange vessels (box 1) which was kept consist. The intensity of fluorescent of the two areas were measured over time. The fluorescent intensity ratio of box 1 over box 2 was plotted against time (Figure 1B) to obtain the slope of the line which leaded to the change in fluorescence intensity over the changing time (solute flux). The equation $P=\Delta I_f/\ \Delta t\ /(\Delta C \times A)$ adapted from Fick's Law was used to obtain the permeability of the vessel in order to assist the observation of DR. Where P is the permeability, $\Delta I_f/\ \Delta t$ is the Solute flux which is the change in fluorescence intensity (box 1 over box 2) over the change in time, ΔC is the difference between intra and extravascular intensity and A is the relative area of vessel to box 1. This described method of using solute flux to calculate apparent solute permeability was obtained from Dave Bates at the University of Nottingham and used with his permission [2]. This equation is still currently under development so Gamez [2] needs to collect enough data of the solute flux to assist the development of this equation.

The current difficulty of this method is all the data points in the fluorescent intensity ratio versus time graph were obtained manually. The position of the selected areas of the capillary (box 1 & box 2) had to be handpicked and adjusted manually for each of the data points due to the movement of the camera caused by the heartbeat of the mice and their eyeball motion. The movement of the camera also caused blurry and unsharp images to appear in the videos which led to inaccurate fluorescent intensity. Therefore, inaccurate data points and anomalies could appear in the graphs. Moreover, this data capturing process was extremely time consuming and repetitive.

The aim of this project is to write a piece of software base on openCV with python and some images and data processing techniques like K-means clustering to allow automated but intelligent data capture from these videos to save researchers time and effort from the data capturing process. More advanced techniques like edge detection will also be implemented to remove blurry and unfocused images out of the data set. This will help to further improve the accuracy of the data points and the slope by reducing the number of anomalies in the graphs. Using camera with higher shutter speed and higher light sensitivity is a very effective way to eliminate blurry frames in the videos but this project has no access to the video capturing process and a large number of videos have already been produced with blurry frames. Therefore, this software will be crucial to automatically process and capture data for all these videos with blurry frames.

1.2 Objectives

1. Create, optimize and evaluate a piece of code based on K-means clustering to segment the exchange and the large vessels group for every frames in the FA videos, then automatically generate a graph of the ratio of the fluorescent intensity in exchange vessel group over large vessel group against time and return the slope of the graph

2. Create, optimize and evaluate a piece of code based on edge detection technique to identify and remove blurry and low-quality FA images out of the data set.

3. Design and implement a graphical user interface (GUI) to combine all the code and package this piece of code into a piece of software that is executable on Microsoft Window operating system.

Chapter 2: Background and Context

2.1 Previous Works of Computer Vision on Fluorescein Angiography:

With the rapid development of advanced computing techniques and methods of biomedical images analysis, eye disease detection by image processing software has been widely used as one of the essential tools by ophthalmologists. In particular, computer vision-based methods are growing rapidly in the field of medical images analysis and are appropriate to advance ophthalmology. Computed image analyses of fundus images could be broadly classified into two big classes [5]. The first type was mainly focusing on techniques of pattern-matching or feature-extraction for the detection and quantification of certain (usually morphologically expressed) pathologies [5]. All these methods relied on structural relationships detection within an image which often included preprocessing steps like averaging out all low-frequency intensity variations within an image to let the global image processing operators, usually the morphological and thresholding operators to run reliably. The second class encompassed on techniques that preserved and used the quantitative relationships within and between images [5]. This is the type of technique this project will be focusing on. This class of technique is particularly applicable to images captured during the clinical procedure called fluorescein angiography (FA). FA is the current top standard technique for retinal vasculature visualization in vivo that has been used for over 30 years to assess choriorentinal disorders [6].

Many researches on computational FA images analysis were done since 1980s. Baudoin et al. [7] proposed a computerized method using the concepts of mathematical morphology to detect and extract microaneurysms particles from fluorescein angiograms by performing different "top-hat transformations". Jagoe et al. [8] demonstrated a computer algorithm which could combine digitized images of fluorescein angiograms sequence of the retinal microcirculation to generate a composite and color image to illustrate circulation at all points in the vascular network. Martinez-Perez et al. [9] suggested using an automated segmentation algorithm for retinal blood vessels from both red-free and fluorescein images to measure the change in morphology of arterioles and venules as diagnostic value for diseases like diabetic retinopathy. Zheng et al. [10] came up with an automated computerized segmentation technique based on Li et al. level-set method that could characterize the size and contour of the foveal avascular zone in diabetic maculopathy by processing fundus fluorescein angiography images from patients with different grades of diabetic retinopathy. Zhao et al. [11] presented a framework involved a state-of-the-art segmentation, geometrical analysis and a trained AdaBoost classifier with weighted cost coefficient to detect intravascular filling defects in fluorescein angiogram images. These researches were mainly focusing on different computerized methods for extraction of variety of quantitative information like morphology, size or contour to detect or segment a variety of microvascular complications like microaneurysms out of the FA images.

Other than quantitative information extraction from FA images, most importantly, FA is the only imaging modality commonly used in DR that can use the visualization of leakage and pooling to provide information on vascular flow and vessel permeability over time [6]. FA images were therefore widely used to detect microvascular complications like the retina capillaries leakage to assist management and diagnoses of wide range of retinal diseases. Phillips et al. [12][13] suggested the very first technique to use digital fundus fluorescein angiograms to detect and quantify macular leakage. Frames from fundus fluorescein angiographic sequence were manually selected to be digitized and a pixel-by-pixel basis was used to observe the rate of change in fluorescence to detect and quantify areas of leakage. Martínez-Costa et al. [14] presented a computer-aided method for macular leakage segmentation. The pixels with a high increment in gray level within the closest area to the foveal center were detected instead of the absolute values of hyperfluorescence to segment the leakage areas which helped to avoid false positives due to poor pigmentation in some patients. Cree et al. [5] proposed an algorithm to restore fluorescein angiographic retinal images that have significant degradation of images due to uneven illumination or occluded optical pathways to allow quantitative

computation could be performed reliably. The restoration process enabled a simplistic macular leakage detection algorithm to run acceptably on a sequence of angiographs. Rabbani et al. [15] proposed an automatic MATLAB-based algorithm for segmenting leakage area in FA images with diabetic macular edema. The algorithm used nonrigid registration of FA frames to obtain early FA and late FA images and subtracted them to remove background artifacts. Follow up with post-processing included detection and inpainting the vessels, and thresholding to obtain the initial contour of the active contour. The Chan-Vese algorithm was used to extract the leakage in region of interest. Zhao et al. [16] developed a framework that could automatically detect large focal, punctate focal, and vessel segment leakage, and validated it on images from patients with malarial retinopathy. The framework involved three steps, vessel segmentation, saliency feature generation and leakage detection. It scored over 80% sensitivity for all three types of leakage detection when compared to manual annotation by expert human observers. Ehlers et al. [17] presented a novel fully automated algorithm which segmented microaneurysms and vascular leakage area in native and dewarped Ultra-widefield fluorescein angiography (UWFA). UWFA is a new technology that improve the vulnerable 30 to 60 degrees narrow field of view from traditional fundus camera. UWFA provides a 160 degrees' field of view which enable the FA images to capture the periphery and far periphery of the retina at the same time and makes any comparison between sections of the retina to be exact in time [18]. This innovative technology allowed Ehlers et al. [17] to create an algorithm that performed similarly to human grader in MA and leakage segmentation.

2.2 Comparison Between This Project and Previous Works

All the computerized automated retina capillaries leakage detection tool described above shares a common point. They were all design and tested on patients with different stage or type of retinopathy which showed obvious symptoms of the retinopathy. Computer algorithm was just used to automatically segment the leakage area out of the FA images. None of those methods were trying to detect the not visible changes in blood vessel permeability at very early stage of the retinopathy. Gamez [2] had started the study on using the change in intensity of fluorescent overtime based on two manually selected region of interest (ROI) in FA images, in combination with the equation $P=\Delta I_f/\ \Delta t\ /(\Delta C \times A)$adapted from the Fick's law to measure the permeability of the retina vessel in order for early diabetic retinopathy detection and treatment. The only recent research on FA-based computer algorithm for detection of the change in retina vessel permeability is proposed by Serlin et al. [19] presented a novel computer-aided method using a fluorescein angiography-based computer algorithm for quantitative assessment of retinal vessel permeability. Vessel permeability was quantified by observations on the temporal changes of intensity for each pixel in the registered images sequence which were normalized and plotted. Normalization was achieved by dividing the intensity of each pixel at a given image with the mean intensity of the manually selected major arteries exiting the optic disk, representing the arterial input function (AIF). A linear curve was fitted for the change in normalized intensity over time, from which the slope was extracted. This approach for vessel permeability quantification is fairly similar to the approach Gamez [2] is using currently, which both required manual selection on ROI which suffer similar difficulties like the ROI position need to be adjust manually due to the movement of the camera causes by movement of eye ball. Also, both approached on permeability measurement were just computer-aided but not fully automated process. This leaves a gap of study on software for detection of retina capillaries leakage at very early stage of retinopathy base on some automated algorithm to extract the non-visible changes in retina vessels' permeability from FA images sequences. All the computerized FA images analysis algorithm shown above demonstrated a high reproducibility and sensitivity when compared to expert human observers' manual annotation on a variety of microvasculature complications.

2.3 Issue of Fluorescein Angiography:

There are still several issues in FA that needed to be aware of. Firstly, quantification of features on FA is typically not as reproducible compared to other imaging modalities such as OCT [15]. Secondly, it was very difficult to attain an appropriate data set for evaluating the reproducibility of the leakage detection algorithm [15]. It was all because FA imaging is invasive and repeated injection of fluorescein dye for research purposes was not permitted by the Institutional review board (IRB). Also, the performance of those algorithms was heavily relied on the quality of the FA images. Automated retinal microvasculature analysis algorithm for grading DR or other eye disease is not expected to replace the experts in diagnostic of disease, but it inevitably reduces the workload of the experts in processing of the retinal images, quickly analyses the disease progress and recommends for early treatment. This project will be mainly focusing on writing a piece of software that allow automated but intelligently capture the permeability of the retina vessels from the FA videos to save researchers time and effort from the data capturing process.

2.4 Background of K-means Clustering

To obtain the permeability of the retinal vessel, the ratio of mean fluorescence intensity in exchange vessel group over large vessel group is needed. The segmentation between the exchange vessel group and the large vessel group is very important and K-mean clustering is proposed to perform this segmentation. The K-means clustering method has been shown to be effective in producing good clustering results for many practical applications [11]. K-means is also a very popular clustering algorithm that has been used in image segmentation [20]. K-means clustering can partition a data set into K groups automatically. One of the disadvantages of K-means clustering is its high sensitivity to initial starting conditions [21][22]. Therefore, finding the exact points to pick as cluster centers is very important for optimal results, along with the optimization of initialization, this is one of the crucial objectives of this project. Also, a direct algorithm of k-means clustering method needs time proportional to the product of number of patterns and number of clusters per iteration [23]. This is a very computationally intensive task especially for big datasets [23][24]. The optimization of the K-means algorithms for the speed and stability of the software is another key objective for this project.

2.5 Background Knowledge on Blur Removal with Edge Detection:

Another crucial objective of this project is removing blurry and unfocused images out of the data set to increase the accuracy of the fluorescent intensity and the permeability values. Edge detection is chosen for this task. Edge detection is widely used to detect blur [25]. An edge in an image can be considered as a gradient between neighboring pixels and edge detection is calculating the gradient between neighboring pixels. At an edge, there will be a large change of the pixel value perpendicular to the direction of the edge, but the change will be low in the direction of the edge [26].

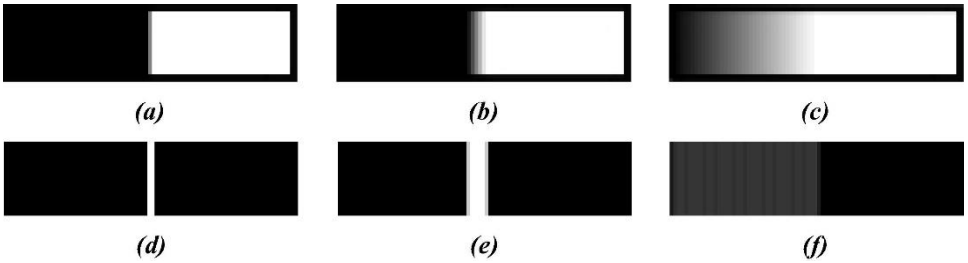


Figure 2 (25): Edges in an image and the results of edge detection. (a) shows a sharp edge between black and white (b) shows a slightly blurred edge (c) shows a very blurred edge with a large transition between black and white. (d) Edge detection result of (a). (e) Edge detection result of (b). (f) Edge detection result of (c).

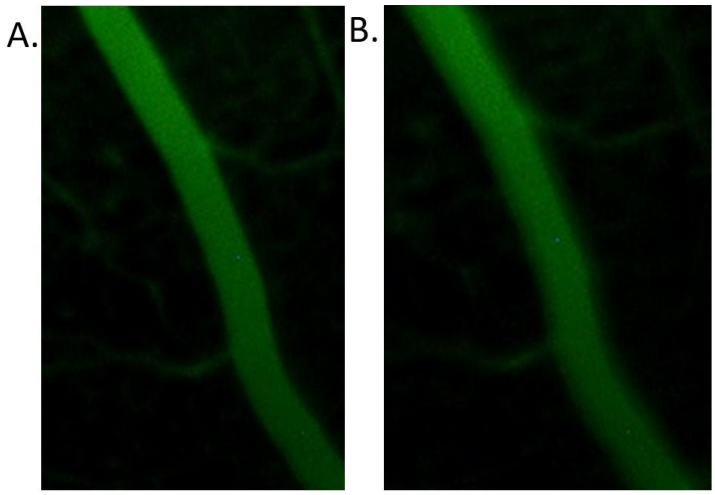


Figure 3:Two FA images from Gamez(2) data set. A. A focused sharp FA images show a shape edge on the large vessel. B. A blurry FA images shows a less defined edge on the large vessel.

The contrast is abrupt between two contrasting colour intensities in sharp images (Figure 2a) and a well-defined result for the edge would be returned from edge detection (Figure 2d). When the blur increases, the contrast decreases and becomes flatter (Figure 2b) and the edge detection result returns a flatter gradient over a larger area (Figure 2e). Edge detection returns barely visible results, or even invisible results (Figure 2f) in case of highly blurred edges with very flat gradients (Figure 2c) [25]. There is a very similar characteristic between sharp and blurry FA images. The blurry FA image showed a softer and less defined edge (Figure 3B) which is like the representation that is shown in figure 2b. Therefore, edge detection is hypothesized for blurry FA images identification.

There are a few edge detection-based blur detection methods can be taken as reference. Ong et al. [27] proposed a method that used the average extent of the slope's spread of all the edges in the image and an additional parameter based on subjective ratings from a small group of human subjects to calculate the blur value. Narvekar and Karam [28] proposed another method that calculated the number of edge pixels in sub-images which were obtained from dividing the image into multiple smaller blocks. Base on the number of edges in the sub-images, a decision was then made whether the sub-images needed further processing, or if enough information was available already.

2.6 Why OpenCV?

This project needed an image processing library to provide built in function for K-means clustering and Sobel edge detection. Due to the lack of prior knowledge on computer vision, a library that is easy to learn and very well documented is crucial. A well-documented library with a less steep learning curve can save time from learning the library and offer more time on experimenting on K-mean clustering and Sobel edge detection. There are few open source computer vision libraries available in the market and OpenCV is one of the most popular one. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million [37]. This assures that there will be huge number of tutorials and documentation available online. It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It provides a very good flexibility for cross language or platform. Therefore, if this project needs to be further developed with a different language or run on different operating system, the flexibility of OpenCV will provide a huge advantage. This library provides the functions for video capturing from video files, image sequences or cameras, K-means clustering algorithm and Sobel edge detection. This library offered the perfect starting point for this project.

2.7 Why Python?

OpenCV supports C++, Python, Java and MATLAB. MATLAB is not free to use so it didn't take into consideration. Although no prior programming experience with Python, Python is still chosen for this project because of numerical reasons. Firstly, Python is open source which makes it free to use and distribute, including for commercial purposes. Python offers very good readability and uncluttered simple-to-learn syntax which helps beginners to learn and utilize this programming language. The wide base of users and active developers has resulted in a rich internet resource and documentation to ease the learning curve further. Python also provides a large standard library which includes areas like GUI design, graph plotting like matplotlib, and machine learning like scikit-learn. Many high use programming tasks have already been included into the standard library which significantly shorten then length of code that needs to be written.

Chapter 3: K-means Clustering Experiment and Exploration:

3.1 Background of Fluorescein Angiography Videos:

Sodium fluorescein angiography (FA) is one of the most effective imaging techniques to be used. Sodium fluorescein is administered intravenously to the diabetic patient, following by an observation on the circulation of the dye thorough recording the retina capillaries [4]. Gamez provided a total of 10 FA video for this project. All videos are 800 X 800 pixels, 2 frames per seconds and in color. Different videos have different durations. All videos showed the capillaries to increase brightness over time (Figure 2).

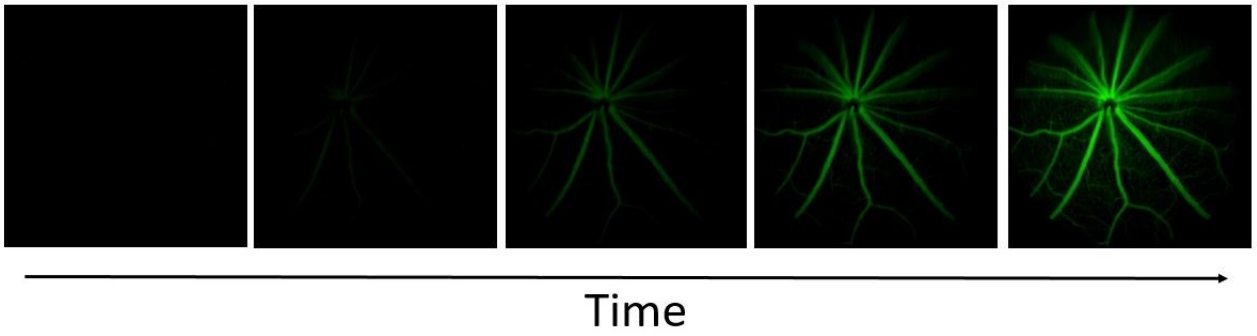


Figure 4: A few selected frame through out one of the FA video to show the change in brightness

Before any analysis can be done on the Fluorescein Angiography (FA) videos, the videos had to be broken down into individual frame first, so that useful information could be extracted from the frame. OpenCV was the chosen library for this task. For the very first trial, a VideoCapture() object was created and the read() method was used to loop along the video frame by frame. All the frame was saved as jpg file into a folder. The idea was to loop along these images one by one. The biggest vulnerability of this approach was the large quantity of frames for each video could use up a lot storage. Therefore, no frames were decided to save down, and all the analysis and data capture would be done during the loop through of the video. Before processing the full video, certain frames were extracted for experiments and further examine.

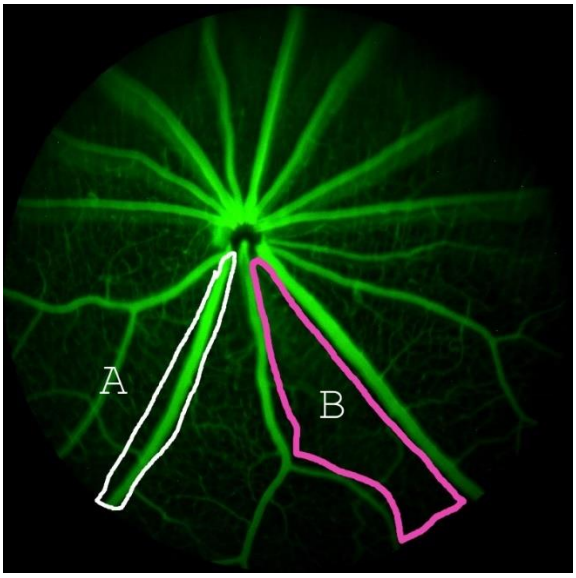


Figure 5: A selected frame from a FA video. Region A is the large vessel region. Region B is the exchange vessel region

One of the main objectives is to distinguish the pixels from the exchange vessel (**EV**) and the large vessel (**LV**) region. The distinctive difference between the EV region and the LV region needed to be identified to distinguish the EV region and the LV region. Base on the human eyes, the visual difference between the EV region and the LV region is the brightness. The LV region is much brighter than the EV region. In order to segment the two blood vessel regions, further knowledge of digital image and a better understanding on the relationship between the EV region and the LV region in the computer aspect is crucial.

3.2 Basic Background of Digital Images:

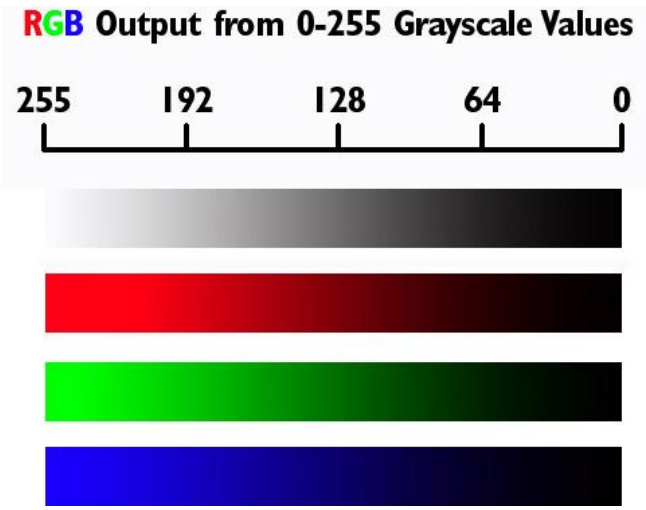


Figure 6[29]: Showed the color and grey scale brightness between 0 to 255 pixel-intensity

Digital images are consisted of a set of numbers or picture elements which concatenate to pixels that are stored as multi-dimensional arrays of numbers in a computer [30]. The individual pixel value represents the brightness of the corresponding point in the image [30]. Most of the time, digital color images are represented by three intensity components in computers, red, green and blue (the RGB model) [30]. Although there are many different color schemes like the CMYK color model (cyan, magenta, yellow and black), this project mainly used the OpenCV BGR color model which is a reordered version of traditional RGB model. The most common format uses 8 bits for each of the three RGB components which can offer brightness levels ranging from 0 to 2^8–1 (255) [30]. 0 to be black in color and 255 is the corresponding brightest red, green or blue color (Figure 6).

3.3 Histogram Study of the Fluorescein Angiography Videos:

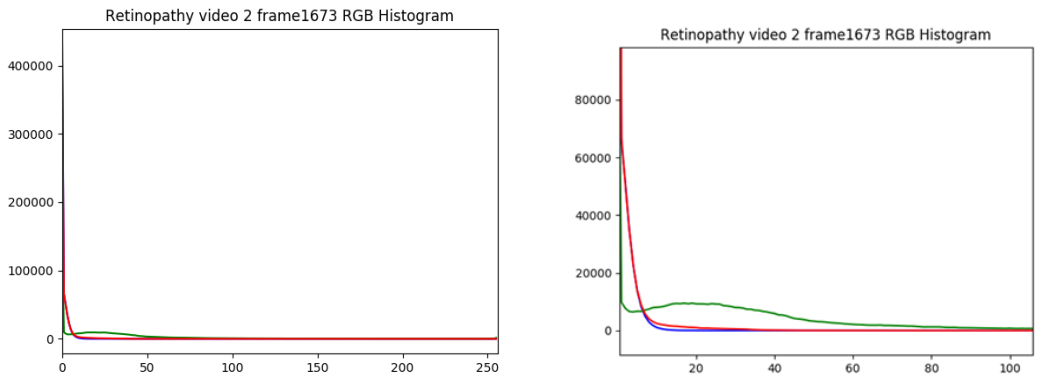


Figure 7: RGB Histogram of frame1673 in video 2, (Left) Full graph, (Right)Zoomed

The histogram of frame 1673 in video 2 was firstly studied. A histogram represents the colors distribution of an image. Figure 7 showed that only the green channel had a variety of pixel value from 0 to around 100. Majority of the pixels have pixel value between 0 to 10, they are known to be the black pixels. Therefore, the frame mainly contains green pixels and a small amount of red pixel as background noise. The green channel was the main focus and the red and blue channel could be ignored. Therefore, converting the frame into grayscale could reduce the complexity of each frame and simplify the image processing process. This grayscale conversion will be done by the OpenCV function cv2.cvtColor(input_image, flag). The flag would be cv2.COLOR_BGR2GRAY in this case.

$$\text{RGB to Gray: } Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \qquad (1)[42]$$

OpenCV uses equation (1) to change RGB or BGR color image into grayscale image. Greyscale image is an array contains the pixel value which represents the brightness of that pixel. These pixel values are stored as 8-bit integers which offers a possible range of values from 0 to 255. 0 is black, 255 is taken to be white and the value in between are different shades of gray (Figure B).

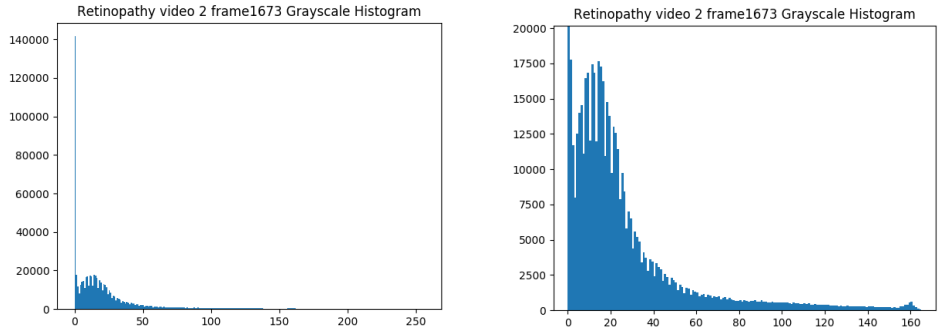


Figure 8: Grayscale Histogram of video frame 1673, (Left)Full graph, (Right)Zoomed

The histogram of the grayscale frame (figure 8) graphically shows the distribution of grayscale pixel values. It demonstrates the distribution of the pixels with different brightness. The left-hand side has the darkest and the right-hand side has the brightest pixel. As shown in figure 8, there is enormous amount of 0 value pixels which are the completely black pixels. This is because of the images from the Optical Coherence Tomography (OCT) only has valid circular image inside the square shape full image. This left four black corners which responsible for the spiking high amount or black pixels. This overcome by cropping out the dark corner.

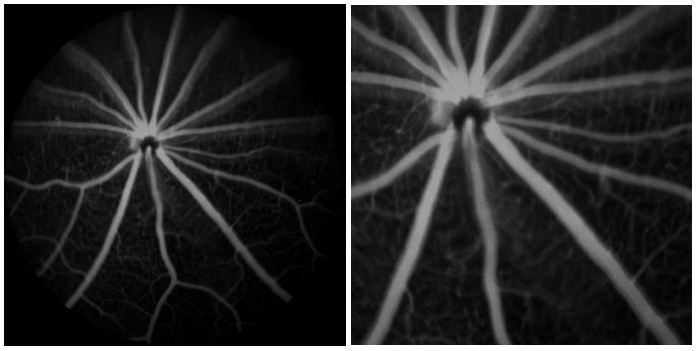


Figure 9: (left) video 2 frame 1673 original frame, (right) video2 frame 1673 cropped 200 to too for both x and y pixel coordinate

As all the video frames were 800 x 800 pixels and for experimental purpose, pixels from 200 to 600 for both x and y pixel coordinate were selected to crop out a 400 x 400 image. A function would be implemented in the final software to allow user to select any ROI of the FA videos.

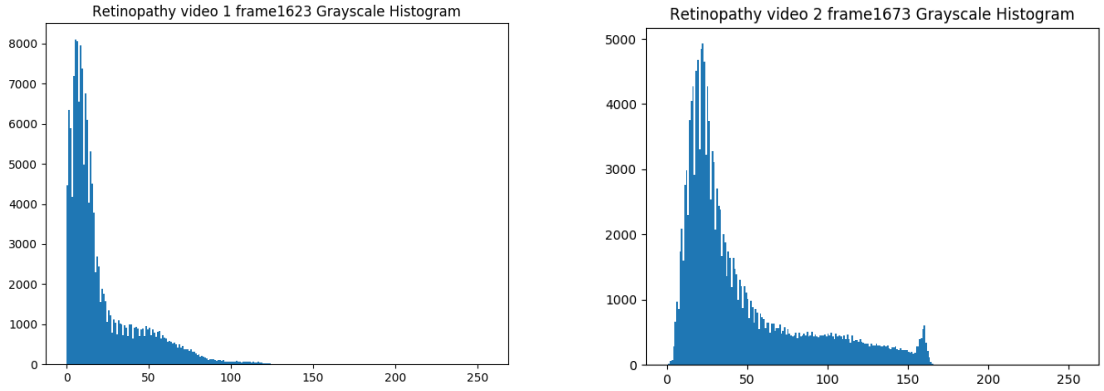


Figure 10: (left) video1 frame 1623 after cropped grayscale histogram, (right) video2 frame 1673 cropped grayscale histogram

Figure 10 demonstrated the histogram of the cropped grayscale image from video 2 frame 1673 (right) and video 1 frame 1623 (left). For video 2 frame 1673 (right), there are two peaks in the histogram. One is between 0 and 50 and the other one is between 150 to 170. As mentioned before, visual difference between the EV region and the LV region is the brightness. The LV region is significantly brighter than EV region. Base on the fact that the pixel value represents the brightness of that pixel, it can be hypothesized that the peak at 0 to 50 pixel-values represent the pixels at the EV region, the peak at 150 to 170 represent the pixels at the LV region. The shape and the features of the histogram may vary with different video. For video 1 frame 1623 (left), there is only one obvious peak between 0 to 25 and many bins spread along to around 125 values. It can also be hypothesized that the peak at 0 to 25 values represent the pixels at the EV region and the bins on the right-hand side represent the pixels at the LV region. It can be hypothesized that the pixels with the lower pixel values are the pixels from the EV region and the pixels with the higher pixel values are the pixels from the LV region. The next step was to separate all the pixels in the frame into two clusters, the high pixel value and low pixel value clusters.

3.4 K-mean clustering theory and implementation:

K-mean clustering from OpenCV was selected to separate the pixels in the frame into two clusters. In the current case, only two clusters, the EV region pixels and the LV region pixel were required so the K was set to 2. The following step was then used by the algorithm to separate data into clusters [43].

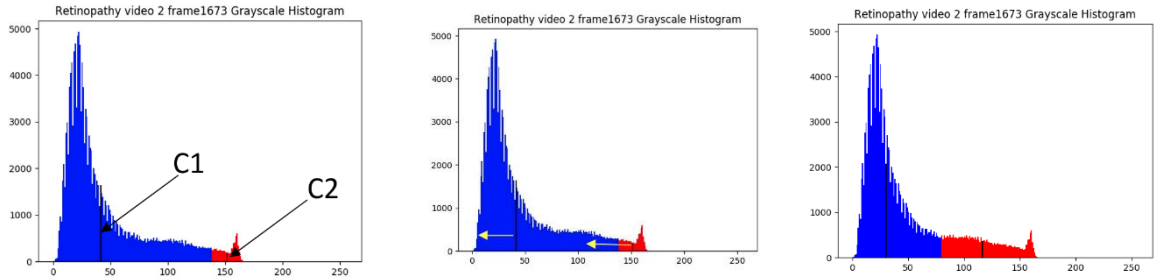


Figure 11: (left)step 1, demonstration of K-mean clustering initial centers selection (middle)step 2, demonstration of K-mean clustering selecting new centers for clusters (right)step 3, demonstration of new resulting clusters after new centers selection,), the images shown are not in true value and not to true scale, it is for demonstration purpose only

Step 1, K numbers of data points were randomly or intentionally picked as centers. In this case, the centers were randomly chosen. As the K was 2, two centers, C1 and C2 were randomly picked. Figure 11 (left) demonstrated that the two black bars were the pixel intensities that were chosen as the two centers.

Step 2, the algorithm calculated the distance between each datapoint to both centers. If the data point was closer to C1, the data was labelled with '0'. If the data point was closer to C2, then it was labelled with '1'. If more centroid was selected, data points could be labelled as '2', '3', etc. As shown in the figure 11 (left), the blue bins were belonging to centroid C1 and the red bins were belonging to the centroid C2.

Step 3, the average of all the data represented by the blue and red bins were calculated separately and that would be the new centers. The C1 and C2 shifted to newly calculated centers (figure 11 (middle)).

$$minimize\left[J = \sum_{all\ blue\ points} distance(C1,\ bluePoints) + \sum_{all\ red\ points} distance(C2,\ redPoints)\right] \quad (2)[43]$$

After C1 and C2 shifted to new centroid, step 2 was performed again to label data with '0' and '1' which leaded to a new set of blue and red bins. Then, step 2 and 3 were iterated until both centers were converged to fixed points. These fixed points were such that the sum of distances between data points and their corresponding centers were minimum. The iteration could also be interrupted depending on the provided criteria like, **maximum number of iteration** or **epsilon** (specific accuracy) was reached. The epsilon is used as a threshold for deciding when K-means has converged. If any center moves more than epsilon units away from its prior position, then the algorithm is not converged, and the algorithm will be run again.

For this K-means clustering implementation experiment, the criteria were set to 10 maximum iteration or reach the accuracy of epsilon = 1.0 to stop the algorithm and return the result. Also, K is 2 and initial centers were randomly picked.

```
#random initial cluster centers
flags = cv2.KMEANS_RANDOM_CENTERS

# criteria is 10 maximum iteration or reach the accuracy of epsilon = 1.0
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

#k is 2, maximum itertaion is 10
compactness, labels, centers = cv2.kmeans(imgArray, 2, None, criteria, 10, flags)
```

K-means clustering function in OpenCV then return three things, compactness, labels and centers. Compactness is the sum of squared distance from each point to their corresponding centers [31]. Labels is the label array that store the cluster label for each of the pixels. It is the same size as the test data and each element marked '0', '1'… etc. Centers is the array of the centers of clusters.

```
#select all the pixel intensity when the label is '0'
clusterA = imgArray[labels==0]

#select all the pixel intensity when the label is '1'
clusterB = imgArray[labels==1]
```

Then based on the labels array, the image data was then split into clusters depending on their labels. Then the clusterA dataset (label 0) was then plotted as histogram in red color, clusterB (label 1) in blue color and the centers in black color. The results of the two selected frames are shown in figure G.

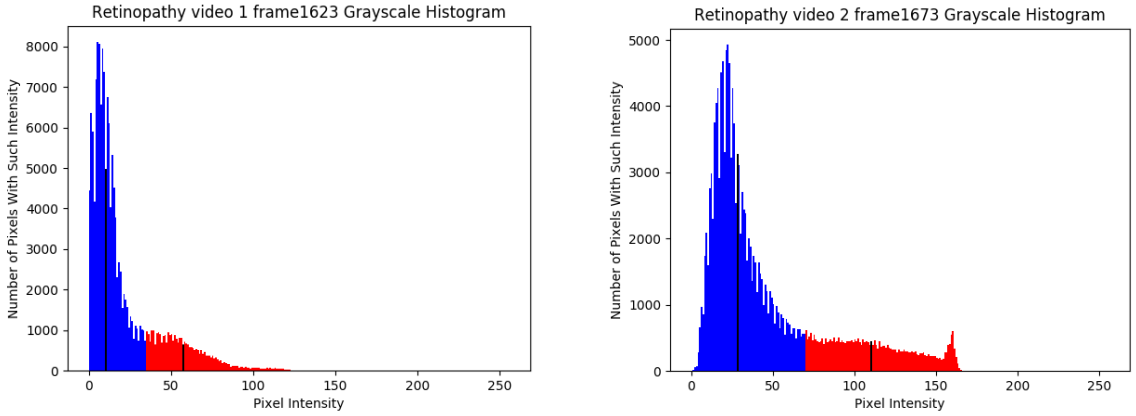


Figure 12: (left)resulting clusters of video1 frame1623 grayscale histogram, (right) resulting cluster of video 2 frame 1673 grayscale histogram

Figure 12 showed the resulting cluster by the K-mean clustering algorithm for the video 1 frame 1623 (left) and video 2 frame1673 (right). Video 1 frame 1623 (left) cluster centers were at 28 and 110 pixel-intensities and video 2 frame 1673 (right) centers were at 10 and 57 pixel-intensities. Both histograms had very similar resulting clusters. One cluster at the lower pixel intensity side and one at the higher pixel intensity side. As mention before, the large vessel group would be brighter which mean higher pixel intensity than the exchange vessel group, so blue cluster (label '1') was hypothesized to be representing the pixels at the exchange vessel group and the red cluster (label '0') representing the large vessel group. In order to confirm this hypothesis, the two cluster were combined and transformed back to an image to display the distribution of the clusters' pixels.

```
#generate an array same size as the label, fill with center[0] or center [1]
#depends on whether the label is 0 or 1
res = centers[labels.flatten()]
#change it back to original image shape
res2 = res.reshape((cropGrayFrame.shape))
```

An array with the same size as the labels array and the image array was filled with the first and second K-means center pixel intensity based on the label '0' or '1'. Then, the array was reshaped back to the original image dimension. This image was only consisted of the pixel intensity of the two K-means centers. Using the video 1 frame 1623 (left) as an example, the label '0' would be represent by 110 pixel-intensity and the label '1' would be represent by 28 pixel-intensity. As 110 is bigger than 28 in pixel value which mean the label '0' cluster would be represent by a brighter region than the label '1' cluster. The label '0' cluster should show the shape of the large vessel.

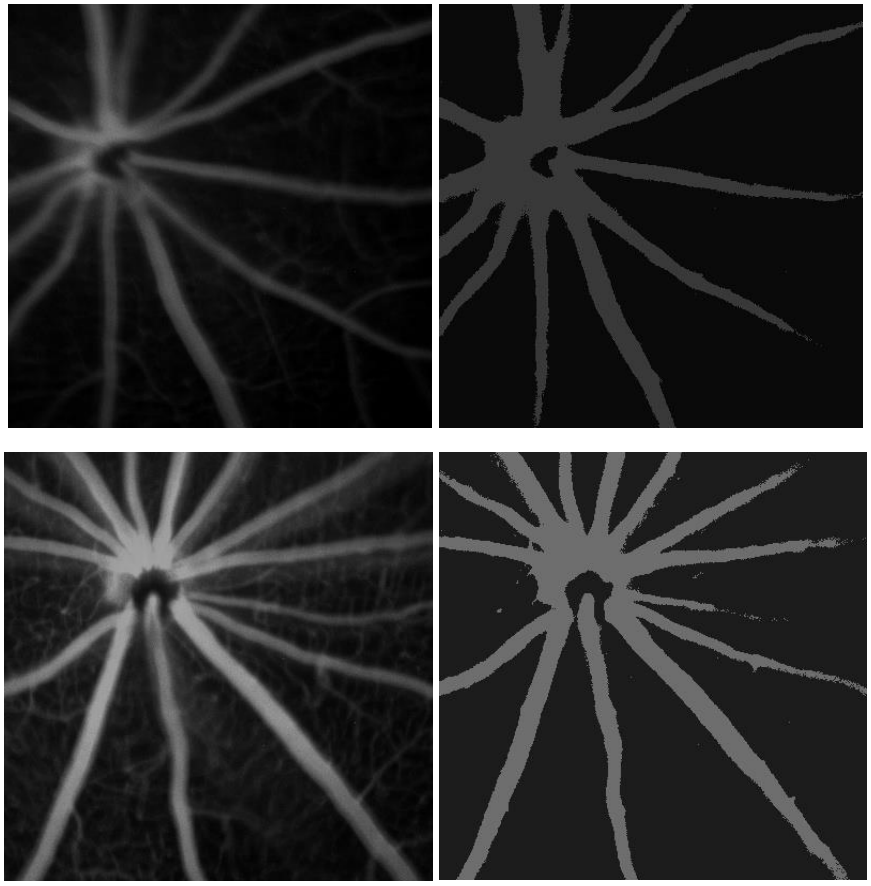


Figure 13: (top left) cropped grayscale video 1 frame 1623 (top right) video 1 frame 1623 K-means clustering resulting clusters (bottom left) cropped grayscale video 2 frame 1673 (bottom right) video 2 frame 1673 K-means clustering resulting clusters

Figure 13 shows the result of the two test frames and the result supported the hypothesis. The two resulting segmented images showed that the brighter pixels from the cluster '0' group into the shape of the large vessel group. This was a very strong piece of evidence to proof that the cluster with higher pixel intensity represented the pixels of the large vessel group and the cluster with lower pixel intensity represented the pixels of the exchange vessel group. The fluorescent intensity could finally be calculated.

Base on the previous work from Gamez [8], the Solute flux ($\Delta I_f/\ \Delta t$) which is the change in fluorescent intensity over the change in time is required for the Fick's Law adapted equation $P=\Delta I_f/\ \Delta t\ /(\Delta C \times A)$ to obtain the permeability of the vessel. According to Gamez [8] previous work, the fluorescent intensity ratio of the exchange vessel group region of interest (ROI) over the fluorescent intensity of large vessel group ROI was required to plot again time to determine the gradient which is the solute flux. In this case, the exchange vessel group ROI was represented by the lower pixel intensity cluster and the ROI of the large vessel group was then represented by the higher pixel intensity cluster. The mean fluorescent intensities of each of the clusters were represented by the two centers' pixel intensity. Graphs of lower pixel intensity center over higher pixel intensity center (Fluorescent Intensity Ratio) versus time graph needed to be plotted to obtain the slope which is the solute flux.

3.5 K-means Parameter Setting Experiment and Evaluation:

There are two variables need to be set for the K-means clustering in OpenCV which are the maximum iteration and the accuracy epsilon. As mention before, the epsilon is used as a threshold for deciding when K-means has converged. If any center moves more than epsilon units away from its prior position, then the algorithm is not converged, and the algorithm will be run again. Experiments needed to be carried out to find the optimal value for these parameters.

One frame from each of the 10 videos were selected for this experiment. The maximum iterations 1 to 15 are experimented. The epsilon 0.1, 0.5, 1 and 2 are also experimented. Radom initial centers were also used. All frames were converted to gray scale and pixel coordination 200 to 600 for both x and y directed are cropped out. Some of the results are shown below (Figure 14), more results are in appendix (1).

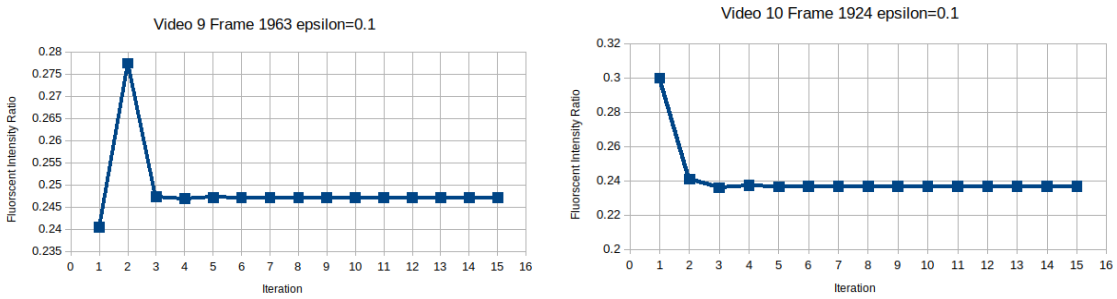


Figure 14: Fluorscent Intensity Ratio vs Maximum Iteration graphs when epsilon = 0.1 for 10 selected frames from the 10 videos

Firstly, all the graphs for all three epsilon values showed the same fluctuation on the fluorescent intensity ratio (FIR) when the iteration below 7. The FIR firstly showed a sharp increase and followed up with a steep drop to a lower value when the iteration is below 7. Only video 4 and 8 showed no sign of falling after the sharp increase but the FIR was still stable at a value. This suggested that the iteration should not be set below 7.

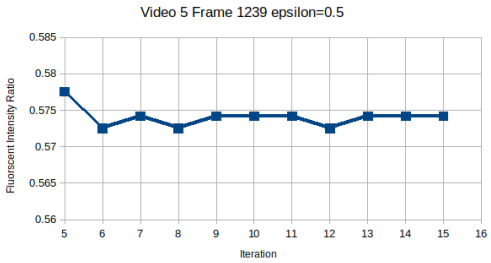


Figure 15: Video 5 frame 1239 Fluorscent Intensity Ratio vs Maximum Iteration graphs with Epsilon=0.5, Zoom in 5 to 15 maximum iterations

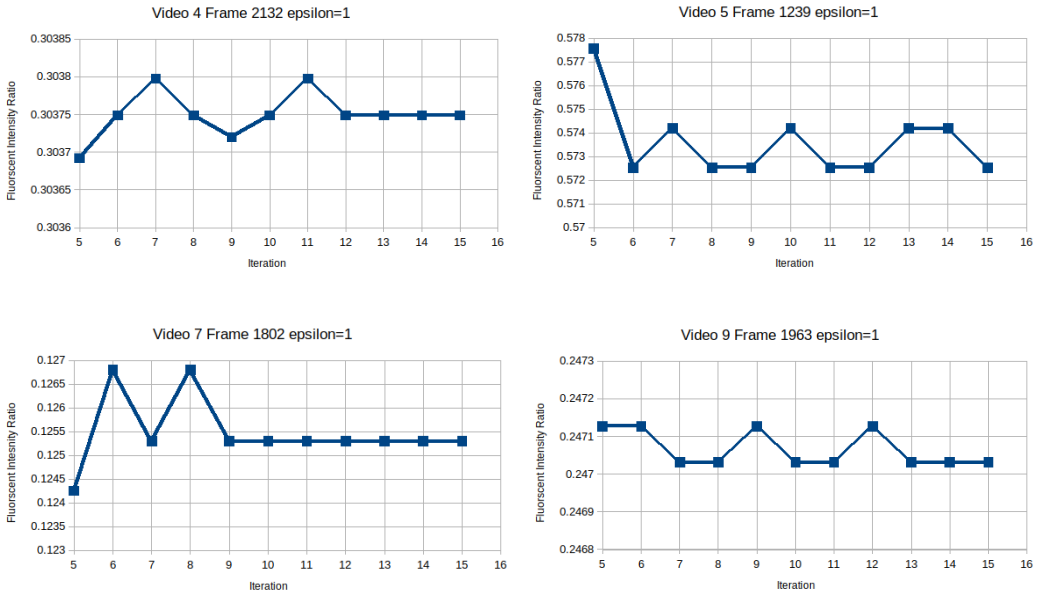


Figure 16: Video 4, 5, 7 and 9 selected frames' Fluorscent Intensity Ratio vs Maximum Iteration graphs with Epsilon=1, Zoom in 5 to 15 maximum iterations

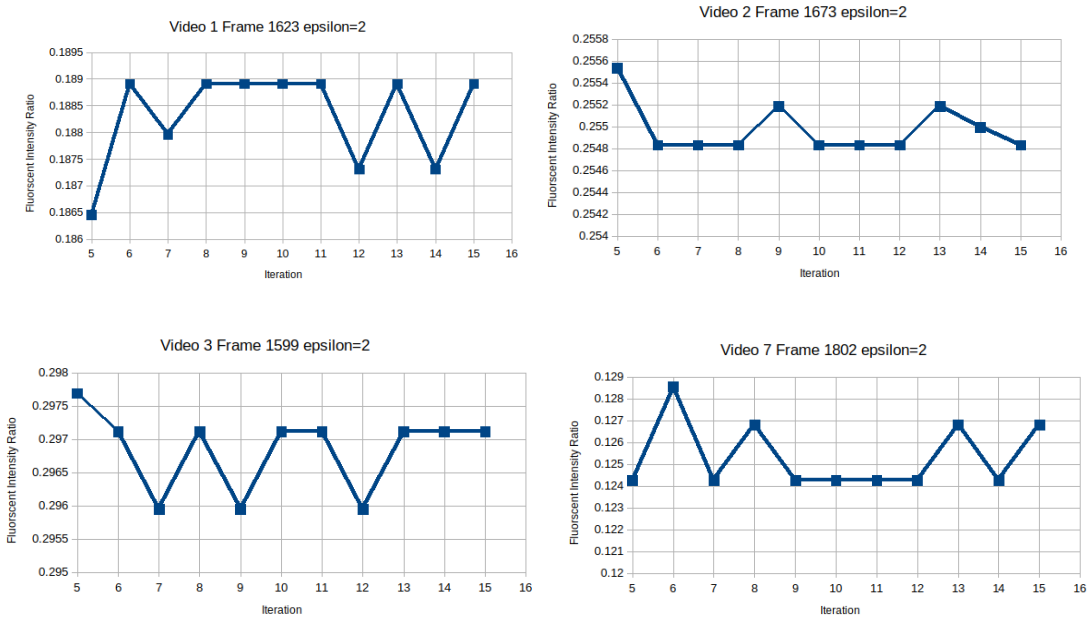


Figure 17: Video 1, 2, 3 and 7 selected frames' Fluorscent Intensity Ratio vs Maximum Iteration graphs with Epsilon=2, Zoom in 5 to 15 maximum iterations

When the epsilon is equal to 0.1 (Figure 14), all the graphs showed that the FIR will be stable when the iteration is 7 or above. When the epsilon increased to 0.5, most of the graphs showed very similar behavior as the graph with 0.1 epsilon except video 5. Figure 15 shows a zoom in version of the video 5 graph, it shows that the fluorescent intensity ratio can fluctuate even when the iteration is larger than 7. The fluctuation was much more minor comparing to the fluctuation when epsilon is 2. When the epsilon value is 1, graphs of video 4, 5, 7, 9 the fluorescent intensity tended to fluctuate even when the iteration is 7 (Figure 16). Graphs of video 1, 2, 3 and 7 showed a more vigorous fluctuation on fluorescent intensity when the epsilon is 2 (Figure 17). This concludes that only 0.1 epsilon can offer stable FIR value. When the epsilon is above 0.1, different frames may show fluctuation on fluorescent intensity ratio.

The iteration should be as low as possible to speed up the run time of the program. As all the video showed a stable FIR value with 7 or more iterations when the epsilon is 0.1, 7 should be the lowest iteration to get the most stable FIR. In conclusion, 0.1 is a very promising number for the epsilon and 7 seemed to be the smallest number that can be set for the maximum iteration to obtain the stable fluorescent intensity ratio.

3.6 Fluorescent Intensity Ratio vs Time Graphs and Evaluation:

After the experimenting on the parameters of the OpenCV K-means clustering algorithm, the videos were looped through every frames of the video to perform K-means clustering to obtain the two clusters' centers and calculate the fluorescent intensity ratio. As mention before, the video 7 showed a vigorous fluctuation on the FIR when the epsilon is above 0.1 but stable at 0.1 epsilon. Therefore, it is the perfect example to show how the number of iteration and epsilon affect the fluorescent intensity ratio vs time graph. Iteration of 1, 3, 5, 7 and 9 were tested while keeping the epsilon to 0.1.

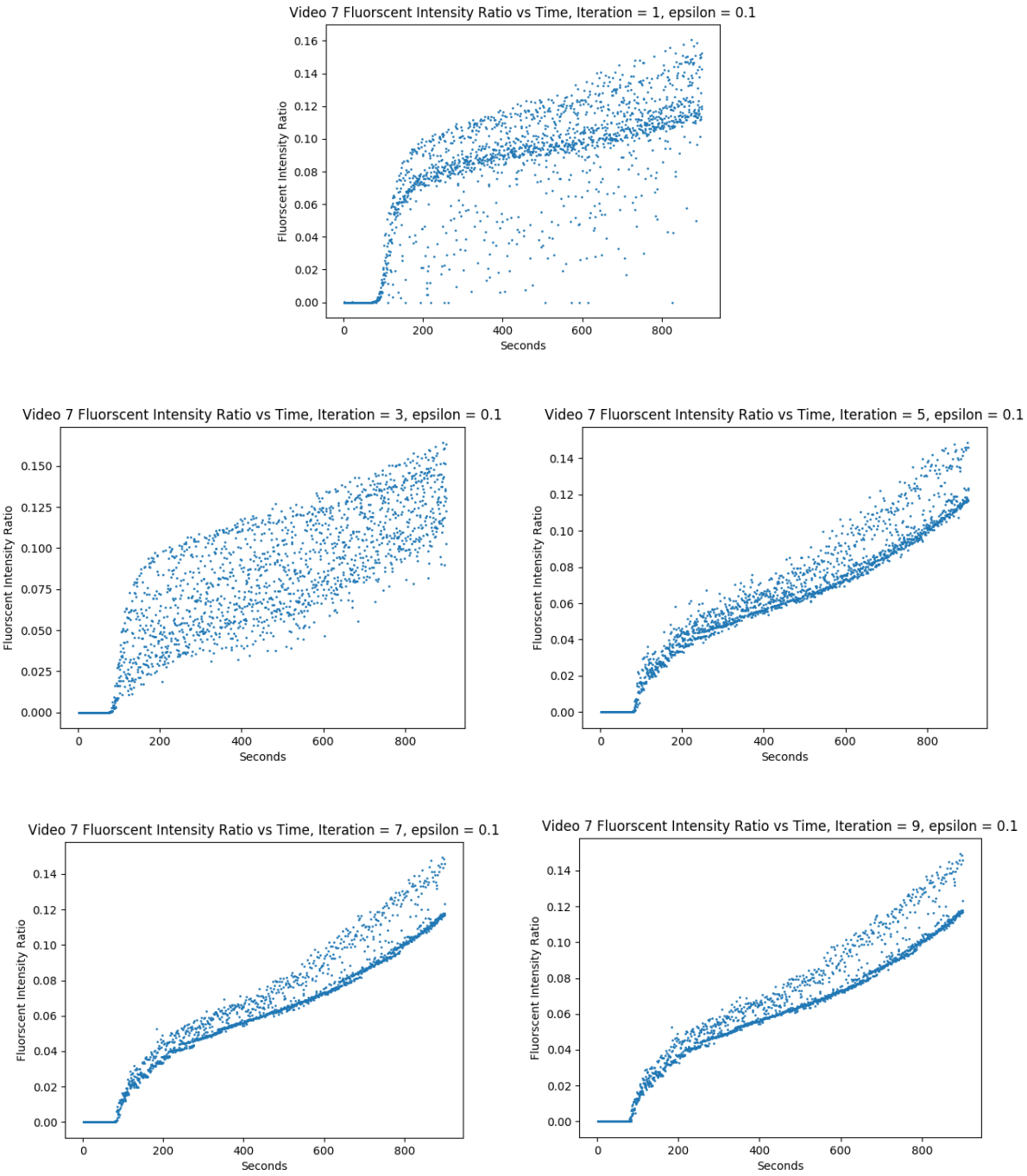


Figure 18: Video 7 Fluorscent Itensity Ratio vs Time graph for maximum iteration of 1, 3, 5,7 and 9 with epsilon 0.1.

Figure 18 shows that the data points were very spread out and noisy when the maximum iteration was 1 and 3. The data points gathered up and showed a clearer trend when the maximum iteration was 5 or above. The data points from the graph with maximum iteration of 5 were still more spread out and noisy compare to the graph with maximum iteration of 7. There was no visible difference between the graph with maximum iteration of 7 and 9. This further proofed the assumption that the maximum iteration shouldn't be set below 7. Otherwise, the FIR vs time graph will be very noisy and out of shape which will affect the accuracy of the gradient. Although there was no visible difference when the maximum iteration was increase from 7 to 9, it cannot assure that there is no imperceptible difference which may affect the accuracy of the gradient of this graph. Therefore, further test needed to be carried out on how the maximum iteration above 7 will affect the gradient of the graph.

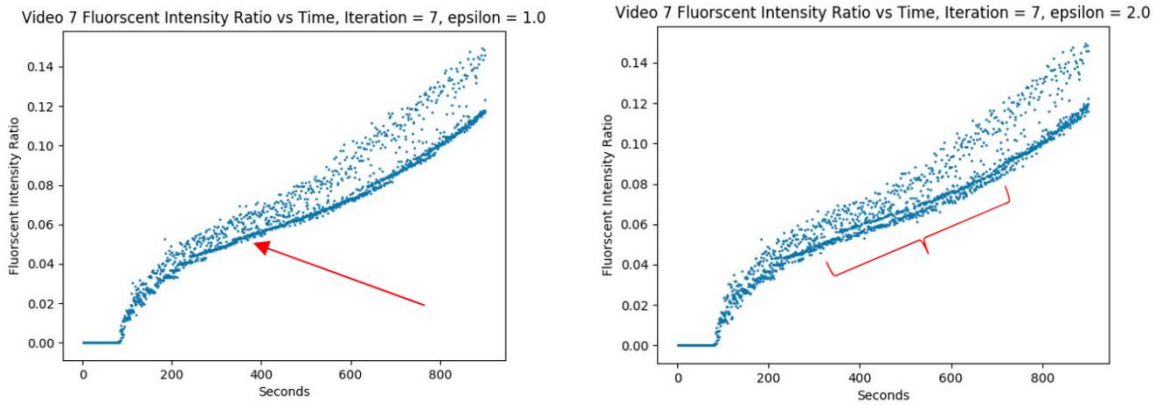


Figure 19: (left) video 7 fluorscent intensity ratio vs time with maximum iteration 7 and epsilon 1, (right) video 7 fluorscent intensity ration vs time with maximum iteration 7 and epsilon 2

The difference is not obvious when the epsilon increased from 0.1 to 1. The red arrow in figure 19 (left) is pointing at a region show extra data point that cannot be seen when the epsilon is 0.1. The graph showed an obvious difference between the graph with epsilon 2 and 0.1. The red bracket in figure 19 (right) highlighted the region with a more spread out data points compared with the graph with 0.1 epsilon. This shows increment of epsilon can increase noise and affect the shape of the FIR vs time graph. Therefore, the epsilon should be kept at 0.1 to prevent the fluctuation of the FIR for most frames of the videos.

The resulting FIR versus time graphs for all the 10 videos provided by Gamez [8] are shown below (Figure 20). All the condition remained constant for consistency, the criteria for K-means were set to 7 maximum iteration or reach the accuracy of epsilon 0.1 to stop the algorithm and return the result. Also, K is 2. Each frame was converted to grayscale and pixels from 200 to 600 for both x and y axis were selected to crop out a 400 x 400 image. Random initial centers were also used.

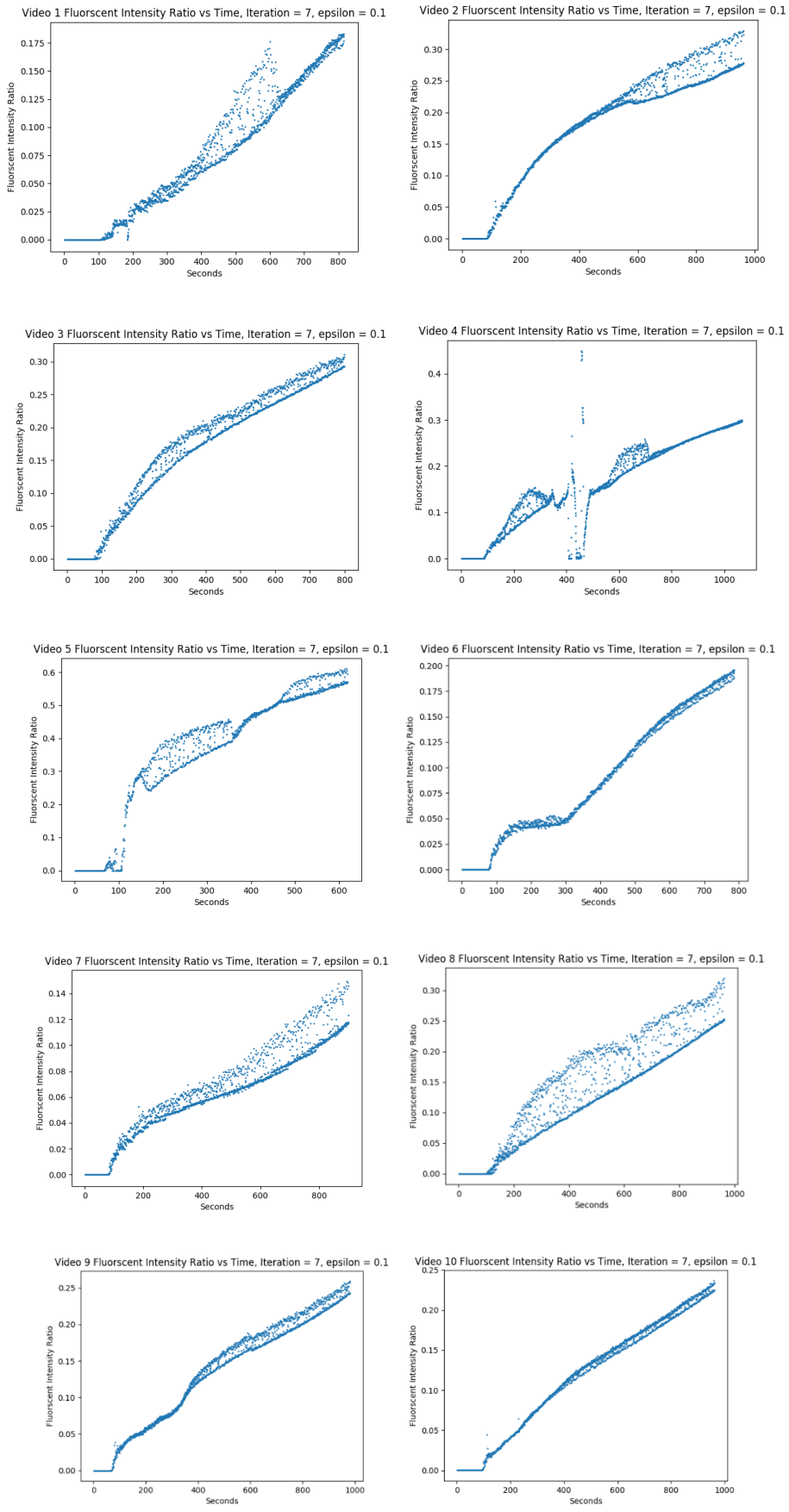


Figure 20: All 10 videos resulting Flluorscent Intesity Ratio vs Time graphs with eps=0.1, max iteration=7, random initial centers

According to figure 20, all the graphs showed the same trend that the fluorescent intensity ratio increases over time. Graphs of the video 3 and 10 had a more obvious curve shape but graph of video 4 and 5 had a more distorted shape. The shape of the graph could be affected by the permeability of the blood vessel itself or affected by the quality of the video frame. The quality of the video frame could be strongly influenced by the amount of the movement of the camera caused by the heartbeat of the mice and their eyeball motion.

Figure 21: video 4 frame 866 ->870 -> 892 ->912 -> 940

Figure 21 is an example of how video quality affected the feature and characteristic of the graph. Figure 21 shows the actual footage of the video 4 when the mouse eyeball moves away from the camera. The frame 892 is complete dark and the fame 912 is completely green which explain the huge range of fluorescent intensity ratio (more than 0.4) between the data points at 400 to 500 seconds.

3.7 How Blurry Frame affect the Fluorescent Intensity Ratio:

Other than the mouse eyeball movement disrupt the image of the blood vessel, most of the graphs in figure I show another obvious defect. The fluorescent ratio fluctuated in most of the graphs which causes some data points to spread out. For example, the graph of video 3 split into two lines which were very close with each other. One line slightly higher fluorescent intensity ratio than the other line. Many other graphs showed this similar defect, for example video 2 ,7, 8, 9 and 10. This defect is hypothesized to be caused by the blurry frames in the video.

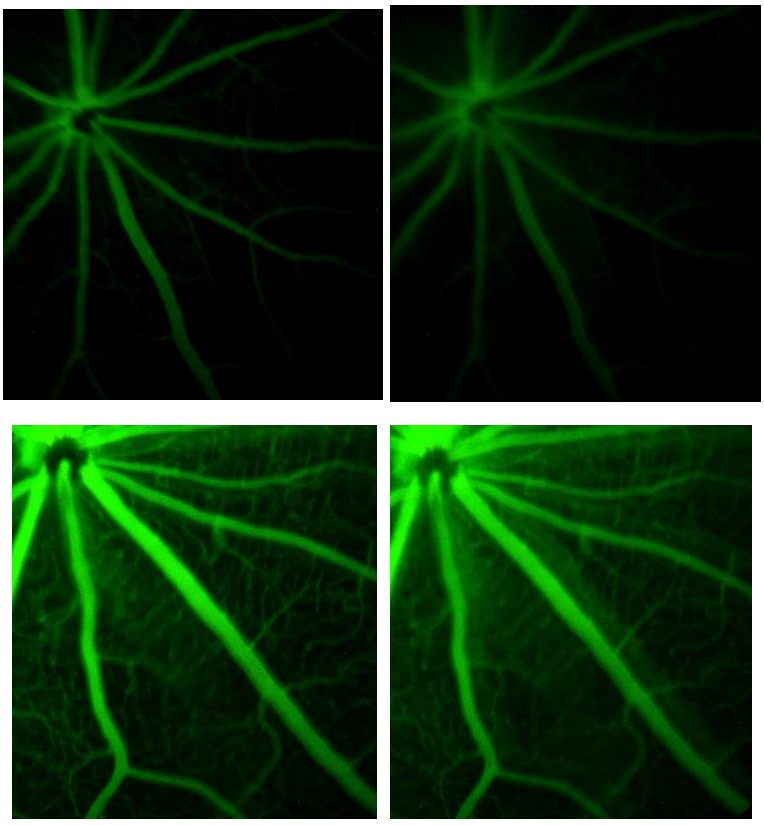


Figure 22: (top left) video 1 frame 1201 which is sharp,(top right) video 1 frame 1202 which is blurry, (bottom left) video 2 frame 1907 which is sharp, (bottom right) video 2 frame 1908 which is blurry

In order to examine how blurry frames would affect the florescent intensity ratio, the K-means centers of four frames, video 1 frame 1201 and frame 1202, video 2 frame 1907 and frame 1908 were compare (Figure 22). They are all cropped with 200 to 600 x and y pixel coordinate. K-means maximum iteration was set to 7 and epsilon was 0.1. The images on the left-hand side are the sharp ones and the one on the right-hand side are the blurry ones. The top right frame has more like a gaussian blur, and the bottom right frame has a motion blur which has a show overlap the large vessel group.

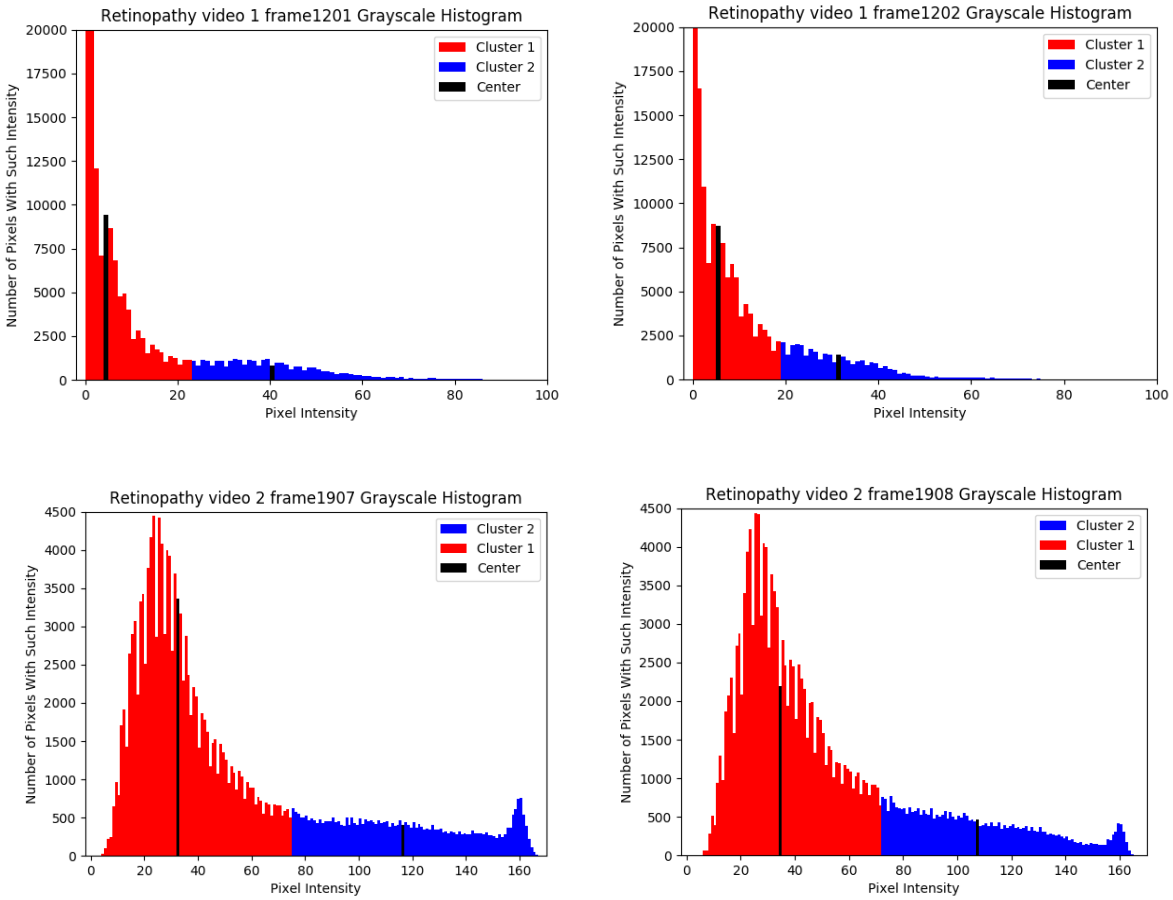


Figure 23: The resulting K-means clustering grayscale histogram of the four selected frames. (left) sharp frames, (right) blurry frames

Video Number	Frame Number	Category	Fluorescent Intensity Ratio
1	1201	sharp	0.1
1	1202	blur	0.16129032258064516
2	1907	sharp	0.27586206896551724
2	1908	blur	0.3177570093457944

Table 1: Show the fluorscent intenisty ratio of the four selected frames

Figure 23 shows all the histogram of the four selected frames. There is visible difference between the sharp frame and blurry frames histograms. For example, the frame 1201 histogram has much lower number of pixels between 10 to 40 pixel-intensity compare to frame 1202 histogram. The frame 1907 histogram has higher peak at the 160 pixel-intensity compare to frame 1908 histogram. Also, frame 1908 histogram has more pixels between 70 to 100 pixel-intensity and frame 1908 histogram has a thinner peak of cluster 1. The selected frames are adjacent frame so the histogram shouldn't be affected by the fluorescent intensity as the time difference is only 0.5 second. The sharp frame 1201 has a lower fluorescent intensity than the blurry frame 1202 (Table 1). The sharp frame 1907 also has lower fluorescent intensity ratio than blurry frame 1908 (table 1). These data shows that sharp frames tend to have lower fluorescent intensity than the blurry frames.

These data support the hypothesis that the blurry frames in the video causes the defects to the fluorescent intensity ratio versus time graph. Therefore, it is crucial to separate out the blurry frame from the video to minimize the number of anomalies in the fluorescent intensity ratio versus time graph. Edge detection was chosen to remove blurry frames out of the video.

3.8 Improvement on K-means Clustering Algorithm's Speed:

Performing K-means clustering algorithm with random initial centers to all frames in the FA video is a time-consuming task due to the number of iterations must be done per frame. The time measured by Python timeit.default_timer for just performing maximum iteration 7 and epsilon 0.1 to all video frames is shown in table 2. This time is unique to the tester computer as different computer hardware can run the code with different speed. Therefore, all time was measured with the same tester computer.

Random Initial Centers, Maximum Iteration 7, Epsilon 0.1										
	Video 1	Video2	Video3	Video4	Video5	Video6	Video7	Video8	Video9	Video 10
Time (Second)	58.7	79.4	66.1	88.4	49.6	62.7	73.3	70.2	80.4	78.7

Table 2: Time to run random initial centers K-means clustering with maximum iteration 7 and epsilon 0.1 to all frames of video 1 to 10

According to table 2, the shortest time is 49.6 seconds and the longest time can go up to 88.4 seconds. This time is quite long. Maximum iteration 7 is the smallest value that can be set to prevent noise and deformation in the fluorescent intensity ratio versus time graph. Therefore, the maximum iteration cannot be further reduced to shorten the run time of K-means clustering. The main reason of the fluorescent intensity ratio versus time graph to be deformed and out of shape when the iteration was below 7 is because of the vigorously fluctuated fluorescent intensity ratio (figure I). This fluctuation of fluorescent intensity ratio is caused by the randomly picked initial centers of the K-means clustering algorithm.

The first 7 iterations of the K-means clustering are moving the random initial centers to the right position to partition the data set into 2 clusters. When the iteration is above 7, the movement of the centers become very small as the partition of the pixels is nearly finished, the algorithm will stop when the centers movement is smaller than epsilon. Rather than randomly selecting initial centers, using the K-means clustering centers of the previous frame as the initial centers of the current frame should be able to reduce the require iterations for stable fluorescent intensity ratio. This is because the difference in brightness and contrast will not differ too much between adjacent frames, the centers of the previous frame should only require small adjustment to fit the current frame to partition the pixels. Therefore, only the very first frame requires the randomly picked initial centers, then the resulting K-means clustering centers will pass onto the next frame as the initial centers and repeat for all the frames (**Reuse centers K-means clustering**). An experiment base on this method was carried out. Maximum iteration 7 and epsilon 0.1 was set for the very first K-means clustering with random initial centers. Then, only reuse centers K-means clustering was performed, maximum iteration 1 was tested on all videos, maximum iteration 3, 5, 7 were tested on video 3 and 7 and epsilon was set to 0.1.

Use Previous Frame's Centers as Current Frame's Initial Centers, Maximum Iteration 1, Epsilon 0.1										
	Video1	Video2	Video3	Video4	Video5	Video6	Video7	Video8	Video9	Video10
Time (Second)	15.2	18.6	15.2	19.8	11.7	14.6	17.6	18.0	18.7	18.0

Table 3: Time to run reuse centers K-means clustering with maximum iteration 1 and epsilon 0.1 to all frames of video 1 to 10

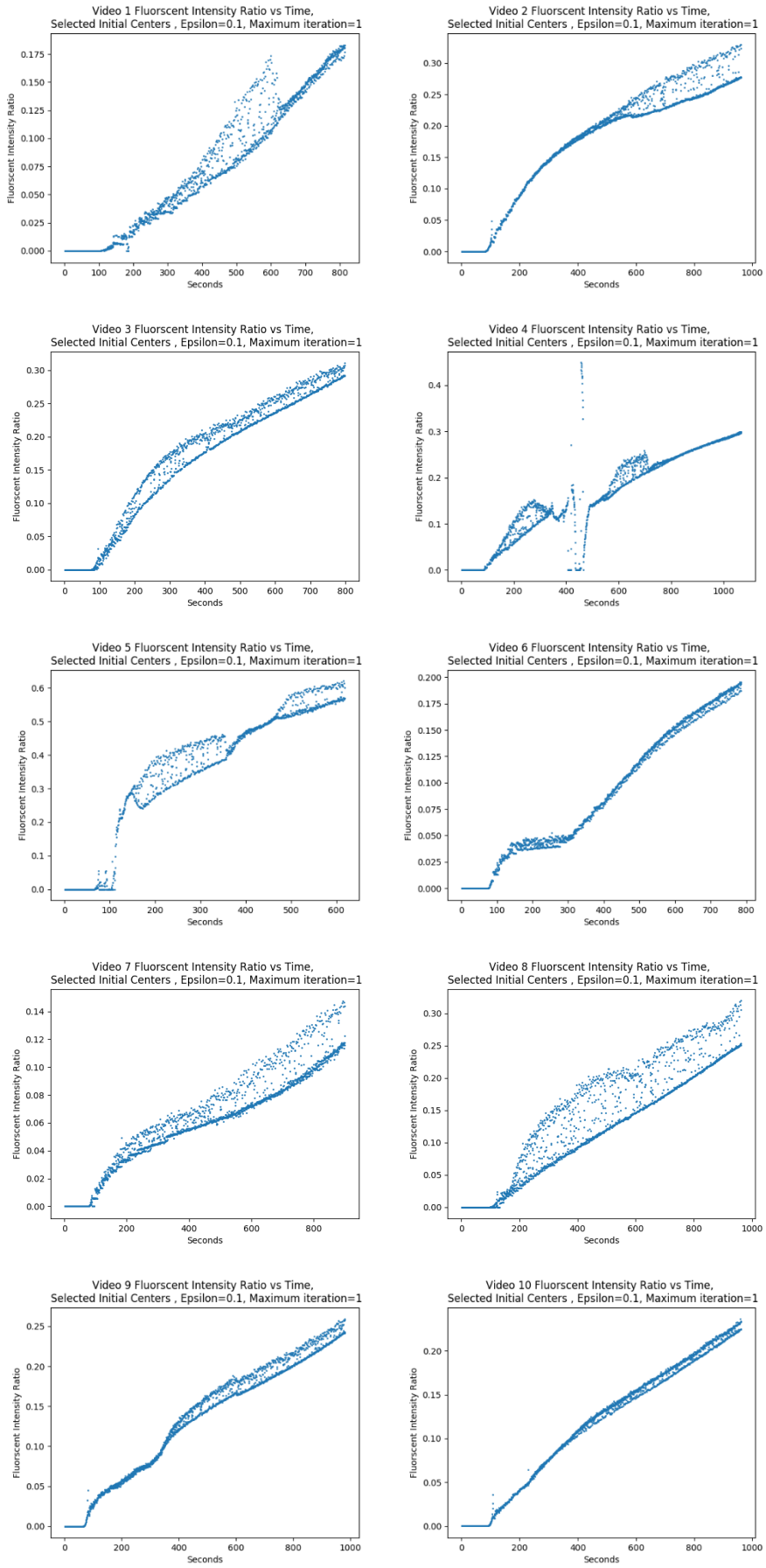


Figure 24: 10 videos' Fluorescent Intensity Ratio vs Time Graphs by reuse center K-means clustering, max iter=1, epsilon=0.1

Figure 24 showed all the fluorescent intensity ratio vs time graph generated by reuse centers K-means clustering, with only 1 maximum iteration and 0.1 epsilon. Table 3 showed the corresponding time to generate these results. The fluorescent intensity ratio vs time graph is very similar to the ones generated by the random initial centers K-means clustering with maximum iteration of 7. No significant visual difference in the shape of the graphs. Unlike the random initial centers K-means clustering that the smallest of maximum iteration for is 7, the maximum iteration can be set to 1 for reuse centers K-means clustering without deformation in the graph.

As maximum iteration of 1 was used, the time difference is tremendous (Table 3). The average time of the 10 videos using random initial centers K-means clustering is 70.8 seconds, the average time for reuse centers K-means clustering was just 16.7 seconds. There is a 4 times difference between them.

Use Previous Frame's Centers as Current Frame's Initial Centers, Epsilon 0.1							
Video 3				Video 7			
Times of Maximum Iteration= 1	Times of Maximum Iteration= 3	Times of Maximum Iteration= 5	Times of Maximum Iteration= 7	Times of Maximum Iteration= 1	Times of Maximum Iteration= 3	Times of Maximum Iteration= 5	Times of Maximum Iteration= 7
15.2	22.3	33.4	56.1	17.6	20.0	37.5	73.8

Table 4: video 3 and 7's times of reuse centers K-means clustering with maximum iteration of 1, 3, 5 and 7

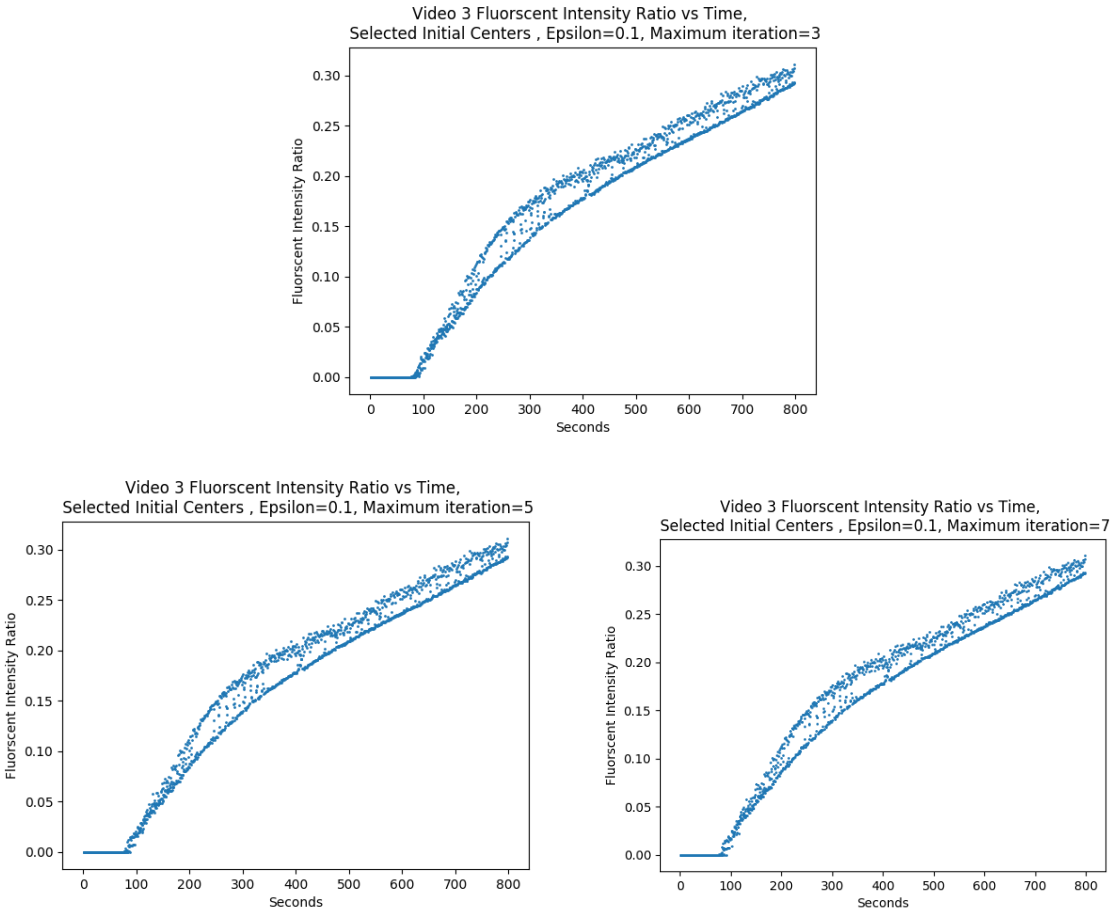


Figure 25: Video 3 fluorescent intensity ratio vs time graps with maximum iteration 3, 5 and 7 and epsilon 0.1

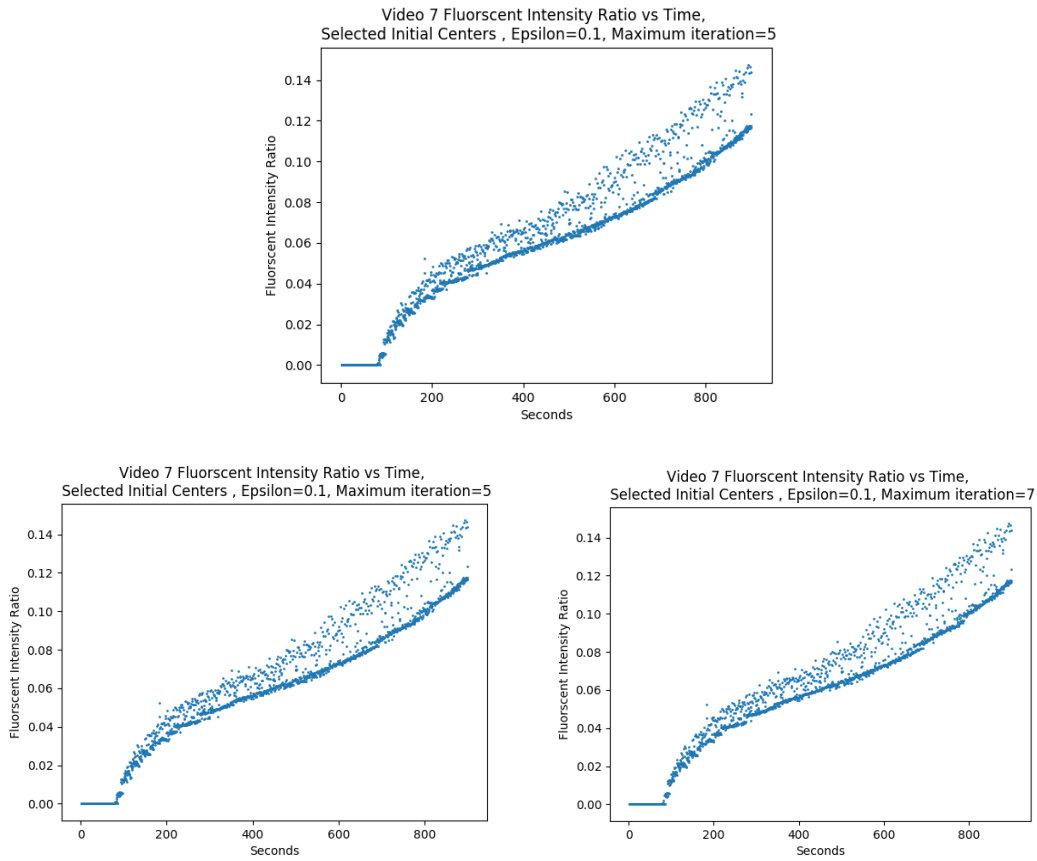


Figure 26: Video 7 fluorescent intensity ratio vs time graps with maximum iteration 3, 5 and 7 and epsilon 0.1

As shown in figure 25 and 26, there is no perceptible difference between the graph with maximum iteration for 1, 3, 5 and 7 for both video 3 and 5. When the maximum iteration was set to 7 for the reuse centers K-means clustering, the time get very close to the one with random initial centers.

Although no visible difference was found in the graphs with different the maximum iteration, imperceptible difference may exist which can affect the accuracy of the gradient of the graphs. Therefore, further test needed to be carried out to discover how the gradient of the FIR vs time graph generated by reuse centers K-means clustering will be affected with different number of maximum iterations. Also, the difference in gradient of the random initial center FIR vs time graph and the reuse centers FIR vs time graph needs to be examined. These experiments can only be carried out after the blurry frame removal code is developed to remove all the outliner data points.

Chapter 4: Blurry Frames Removal:

4.1 Background Knowledge of Blur Removal

4.1.1 Image Convolution:

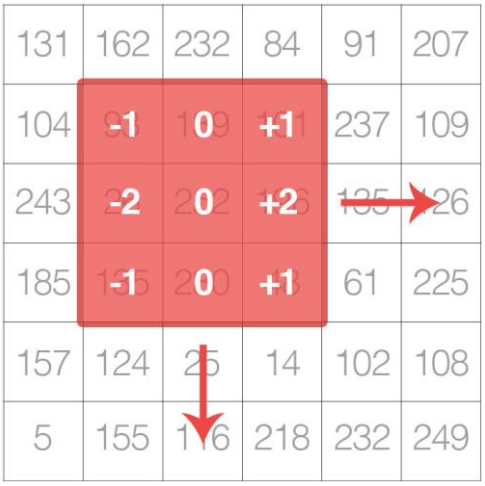


Figure 27 [34]: image mimicking the kernel matrix move on top of an image array

Before working with edge detection filters, a solid understanding on image convolutions is crucial. As mentioned before, digital image is just a multi-dimensional array or matrix that stores all the pixel values. An image is a big matrix and kernel or convolutional matrix as a small matrix that is used to process the image like blurring, sharpening, edge detection etc. This small matrix just lay on top of the big image and move from left to right and top to bottom (Figure 27), applying a mathematical operation at each pixel values of the original image. At each pixel value of the original image, the kernel will examine all the neighbors of the pixels at the center of the kernel, the pixel neighborhood will perform a convolution with the kernel.

1	2	3	x	a	b	c	=	1 x a	2 x b	3 x c
4	5	6		d	e	f		4 x d	5 x e	6 x f
7	8	9		g	h	i		7 x g	8 x h	9 x i

Figure 28 [34]: image mimicking the convolution operation between two matrices, this is not a normal matrix multiplication

The convolution being performed is not a tradition matrix multiplication but a different matrix operation. It takes the element-wise multiplication of the selected image region and the kernel. Figure 28 demonstrates a convolution matrix operation. Finally sum all the values in the resulting matrix to obtain a single output value. This output value will then be stored at the same coordination as the center of the kernel. Figure 29 demonstrate the whole process.

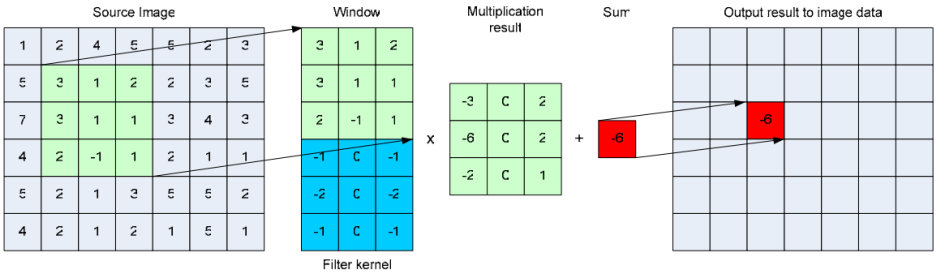


Figure 29[34][44]: Demonstration of the whole process of image convolution

Pre-define kernels are very commonly used to obtain a variety of image processing function. For example, average smoothing, Gaussian smoothing are pre-defined kernels for blurring and Laplacian, Sobel and Canny are examples of pre-defined kernels for edge detection.

4.1.2 Sobel derivative operator:

Sobel derivative operator was chosen to perform edge detection for this project. Sobel derivative operator detects the edges by searching for the maximum and minimum in the first derivative of the image.

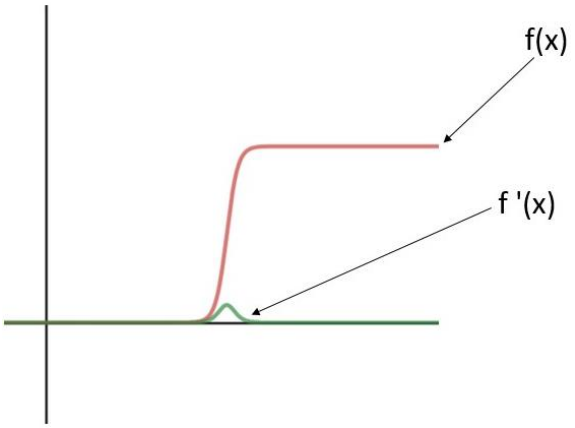


Figure 30: f(x): A sigmoid function mimicking the change of pixel intensity of an edge, f ' (x): the first derivative of f(x)

As mention before, there will be a large change of the pixel intensity perpendicular to the direction of the edge and figure H f(x) is mimicking this large change of pixel value. If the first derivative is taken of f(x), the graph turned into f '(x) in the figure 30.

The first derivative (f '(x)) clearly shows a maximum located at the center of the edge in the original graph. The Sobel operator is basically a discrete differentiation operator. It computes an approximation of first derivative of image intensity at each pixel within the image. The Sobel filter highlight the position of the edge by finding the maximum of the derivative of the image. This method of edge positioning is a characteristic of the "gradient filter" family of edge detection filters which includes the Sobel operator [32].

The first derivative of the image is a vector, which measures how rapid pixel value are changing with distance in the x and y direction [33]. The components of the gradient may be found using the following approximation, equation (3) and (4):

$$\frac{\partial f(x,y)}{\partial x} = \Delta x = \frac{f(x+dx,y) - f(x,y)}{dx} \quad (3)\ [33]$$

$$\frac{\partial f(x,y)}{\partial x} = \Delta y = \frac{f(x,y+dy) - f(x,y)}{dy} \quad (4)\ [33]$$

The *dx* and *dy* measure along the *x* and *y* directions respectively. *dx* and *dy* can be considered as numbers of pixel between two points in discrete images. *dx*=*dy*=1 is the point at which pixel coordinates are (i, j). Therefore, the following equation represent the estimation of the intensity gradient at a pixel at the x and y direction for an image:

$$\Delta x = f(i+1,j) - f(i,j) \quad (5)\ [33]$$

$$\Delta y = f(i,j+1) - f(i,j) \quad (6)\ [33]$$

The magnitude of the intensity gradient estimation is given by the following equation (7):

$$M = \sqrt{\Delta x^2 + \Delta y^2} \quad (7)\ [33]$$

The differential operators in equation 1 and 2 can be represented by the following kernel for convolution, equation (8):

$$\Delta x = \begin{matrix} -1 & 1 \\ 0 & 0 \end{matrix} \qquad \Delta y = \begin{matrix} -1 & 0 \\ 1 & 0 \end{matrix} \quad (8)\ [33]$$

Sobel edge detector normally use a larger kernel size. The advantage of using a large kernel is the effect on the accuracy by noise is reduced by local averaging within the neighborhood pixels of the kernel [33].

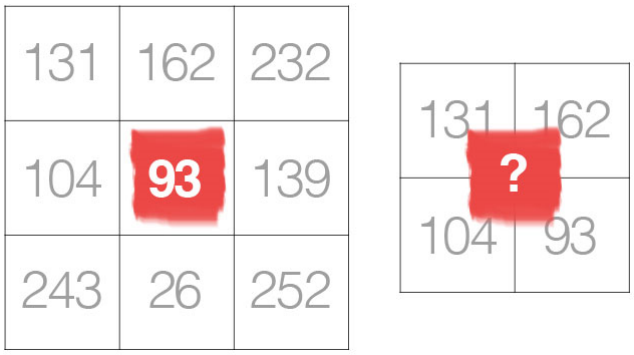


Figure 31 [34]: Demonstrate there is no center pixel if the kernel size is even number.

Also, odd number size of kernels is normally used. This is because the kernel is centered and can therefore provide an estimate gradient that is according to the center pixel inside the kernel (which is the 93 in the figure 31).

$$\Delta x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad \Delta y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

$$\Delta x = \begin{matrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{matrix} \qquad \Delta y = \begin{matrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

Figure 32: (top) x and y direction 5x5 sobel kernel generated by cv2.getDerivKernels(), (bottom) x and y direction of 7x7 sobel kernel generated by cv2.getDerivKernels()

Figure 32 shows the two most commonly used Sobel operator which are the 3 and 5 kernel size. One for the x direction and one for the y direction. These kernels will undergo a convolution operation to the image and return an output image array of the same size and the same number of channels as the input image that only contain the pixel intensity gradient estimation at x or y direction. Therefore, both the x and y direction Sobel operator had to be used to generate an output array with the magnitude of the intensity gradient estimation. Figure J show an example of this convolution operation:

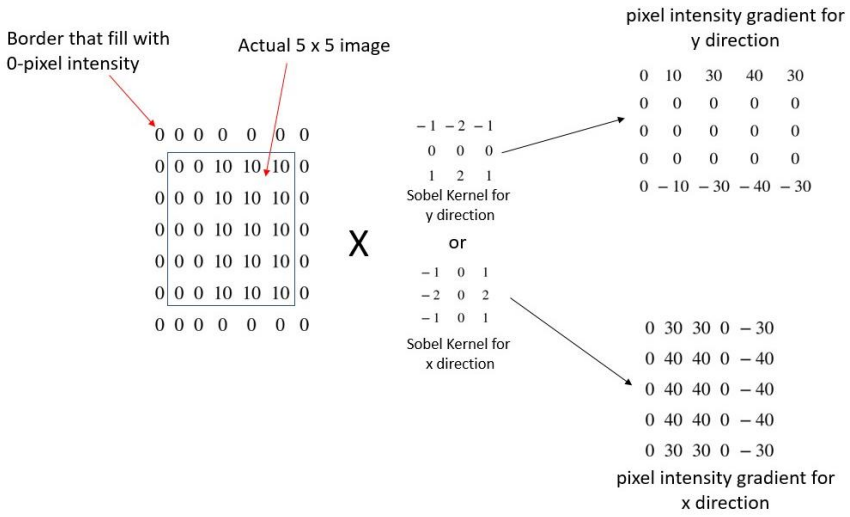


Figure 33: demonstration of the Sobel kernel convolution operation to a 5 x 5 pixel edge image

According to figure 33, the original 5 x 5 pixels image with an edge at the x direction. This original image is surround by a border of 0 intensity pixels. This pixel border is very important. As mention before, odd number size kernels are normally used since the kernel is centered so it can provide an estimate gradient that is according to the center pixel inside the kernel. What if the pixels are at the edge or the corner of the photo, there is no such thing as “center” pixels for these pixels. The decrease in spatial dimension is one of the side effects of applying convolutions to images. In order to make sure the output image to have the same dimensions as the input image, this extra border around the image can help to ensure the output image will match the dimensions of the input image.

The result image array of the x direction Sobel kernel showed that the left side of the edge was filled with positive 30 and 40 values and the right side of the edge was filled with negative 30 and 40 values. The middle column of the is fill with 0 as it is not an edge anymore, it is in between the two edge. The result image array of the y direction Sobel kernel only picked up the top and the bottom edge of the original image, the x direction edge was just filled with zero.

4.2 OpenCV Sobel Edge Detection Experiment and Evaluation:

The Sobel operator from OpenCV is used for edge detection.

```
#Gradient in x direction
grad_x = cv2.Sobel(crop_img, cv2.CV_64F, 1, 0, ksize=5)
# Gradient in y direction
grad_y = cv2.Sobel(crop_img, cv2.CV_64F, 0, 1, ksize=5)
```

cv2.Sobel (src, dst, ddepth, dx, dy, ksize, scale, delta, borderType)

src = input image

dst = output image of the same size and the same number of channels as src

ddepth = output image depth

dx = order of the derivative in x direction

dy = order of the derivative in y direction

ksize = size of the extended Sobel kernel, it must be 1, 3, 5, or 7

scale = optional scale factor for the computed derivative values, by default, no scaling is applied which is 1

delta = optional delta value that is added to the results prior to storing them in dst, by default none is add which is 0

borderType = pixel extrapolation method, by default it is set to cv2.BORDER_DEFAULT

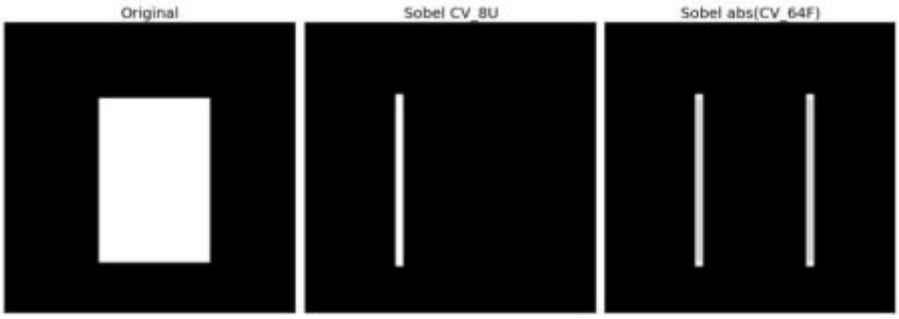


Figure 34 [35]: demonstration the edge detection result of using cv2.CV_8U (middle) and cv2.CV_64F (right)

Here is an experimental attempt on using the OpenCV Sobel operator. Only src is set as crop_img and dst is the same as the src. The ddepth is set to cv2.CV_64F but the input image is just a grayscale np uint8 or cv2.CV_8U. The reason of doing is to make sure the Sobel operator can detect and store all the edges.

Black-to-White transition is interpreted as Positive slope (it has a positive value) and White-to-Black transition is interpreted as a Negative slope (It has negative value). Therefore, if the data is converted to np.uint8 (cv2.CV_8U), all negative slopes are made zero. In simple words, the white-to-black edge will be missed.

In order to detect both edges, better option is to keep the output datatype to some higher forms, like cv2.CV_16S, cv2.CV_64F etc, take its absolute value and then convert back to cv2.CV_8U. Figure 34 shows the difference in results for a horizontal Sobel filter between Sobel using CV_8U and CV_64F.

In order to calculate the gradient in x direction, dx = 1 and dy = 0. For the gradient in y direction, dx=0 and dy=1. In this experimental attempt, ksize = 5. OpenCV support 1, 3, 5, 7 kernel size only. 1 for 3 x 1 or 1x 3 kernel, 3 for 3 x 3, 5 for 5 x 5 and 7 for 7 x 7 kernel. Scale and delta are set to default. The borderType is also set to default which is the cv2.BORDER_DEFAULT. cv2.BORDER_DEFAULT is also cv2.BORDER_REFLECT_101 which will create a border around the image, which is something like a photo frame. cv2.BORDER_DEFAULT create the border following this rule gfedcb |abcdefgh| gfedcba, where the image boundaries are denoted with '|'.

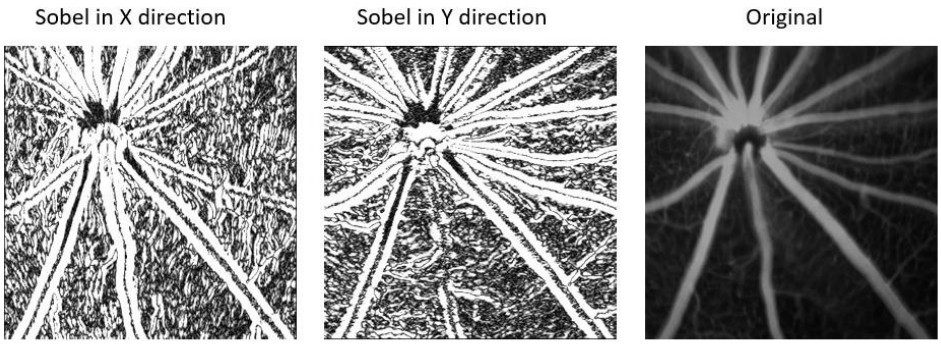


Figure 35:(left) the resulting image of X direction Sobel operator, (middle) the resulting image of Y direction Sobel operator, (right) original image

Figure 35 show the result for X and Y direction size 5 Sobel operator on video 2 frame 1907 at 200 to 600 pixel-coordination for both X and Y direction (400 X 400 pixels). As you can see the white pixel are the edge of the large vessel group. The edge is white means they have higher intensity. This proof that the Sobel operator can identify the edge of the large vessel group. It can pick up some of the exchange vessel group too but not as visible as the large vessel group.

The next step is to find the magnitude of intensity gradient estimation and turn it into a single value that can be used to detect blurriness of images. The magnitude of intensity gradient is based on equation (7). Firstly, square all the values in the X and Y direction Sobel operator resulting image array. Then, sum the X direction value with the corresponding Y direction value. Finally, square root all these sum value individually. This result an array of the magnitude of intensity of gradient estimation. Then, all the value in this array will be sum up into one value and then divided by the total number of pixel (in this case it is 400*400) to create a single value to represent the "**total edge sharpness**" or just "**edge sharpness**" in order to detect blurry images. This process is performed by the following code:

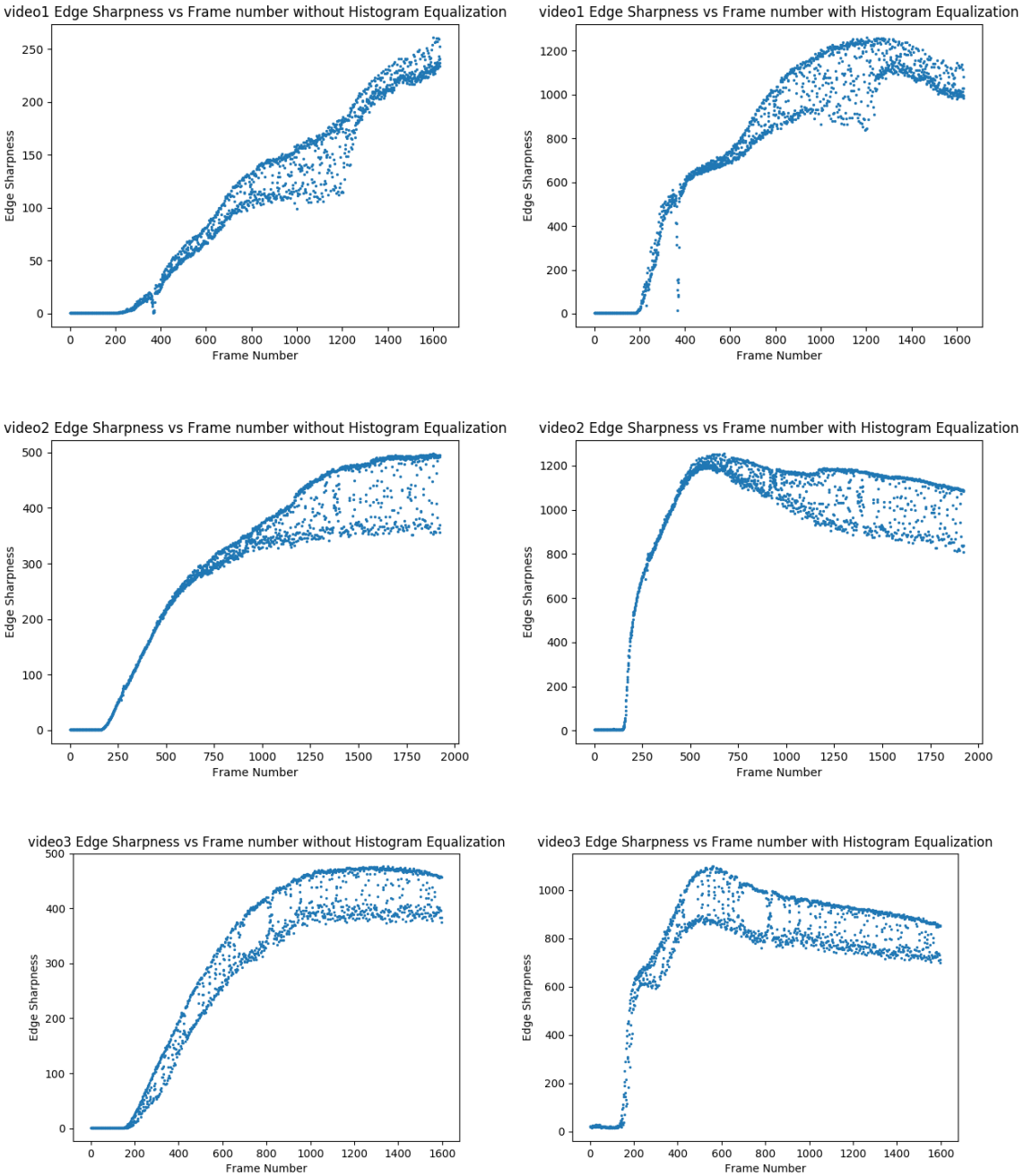
```
#Gradient in x direction
grad_x = cv2.Sobel(crop_img, cv2.CV_64F, 1, 0, ksize=5)
# Gradient in y direction
grad_y = cv2.Sobel(crop_img, cv2.CV_64F, 0, 1, ksize=5)


#Square all the x direction values
grad_x_square = np.square(grad_x.ravel())
#Square all the y direction values
grad_y_square = np.square(grad_y.ravel())

#sum x direction values to the corrosponding y direction values
#square root all the values, then sum all the value into one single value
totalEdge = np.sum(np.sqrt(np.add(grad_x_square, grad_y_square)))/(400*400)
```

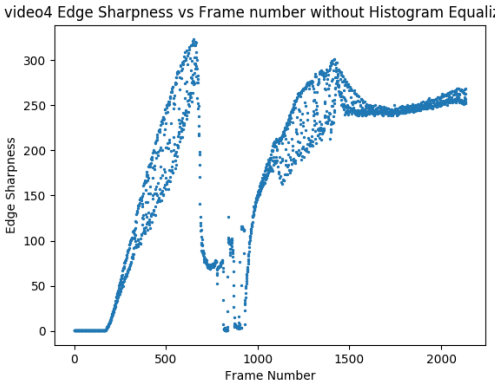
As sharper the image, harder and more defined edges will be showed, there will be a larger change of the pixel value perpendicular to the direction of the edge and the Sobel operator should output larger value or more non-zero value as more edge could be detected. Therefore, the total edge sharpness value will increase, and the Sobel operator output larger value for the pixels at the edge. Therefore, it was hypothesized that sharp frame will have larger edge sharpness value than the blurry frames. The video 1 frame 1201 and frame 1202, the video 2 frame 1907 and frame 1908 (figure K) are tested with the Sobel kernel with a size of 5 and their edge sharpness values are compared. The frame 1201 and 1907 are the shape one and the frame 1907 and 1908 are the blurry one.

Video Number	Frame Number	Category	Edge Sharpness Value
1	1201	sharp	182.78910843434758
1	1202	blur	111.77068675934781
2	1907	sharp	486.1832658186319
2	1908	blur	374.0217158095304

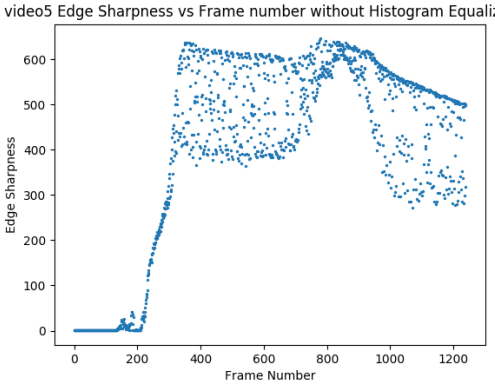
Table 5: Show the edge sharpness value of the four selected frames

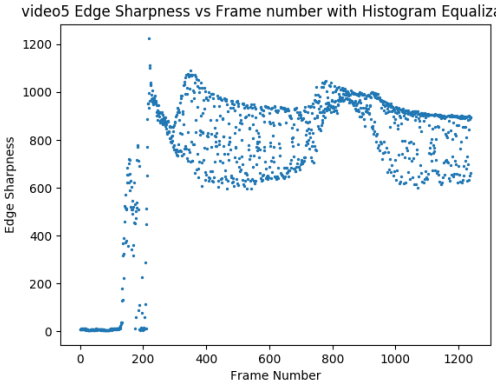
The data shows that the blurry image has much lower edge sharpness value than the sharp image. Now the size 5 Sobel operator is performed onto all frames of the videos to obtain the edge sharpness value and the edge sharpness value vs frame number graphs were plotted. The results are shown below on the Figure 36 (left):

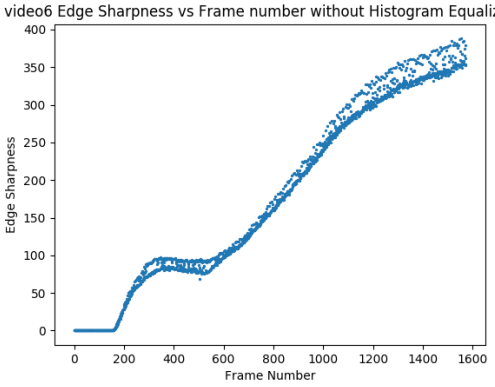


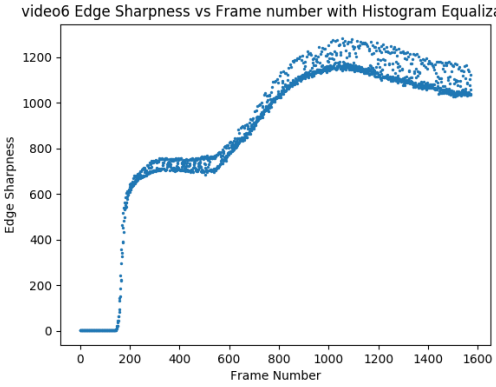

video4 Edge Sharpness vs Frame number without Histogram Equalization
Edge Sharpness
Frame Number

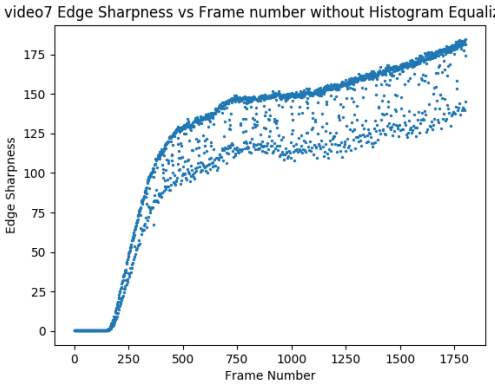

video4 Edge Sharpness vs Frame number with Histogram Equalization
Edge Sharpness
Frame Number

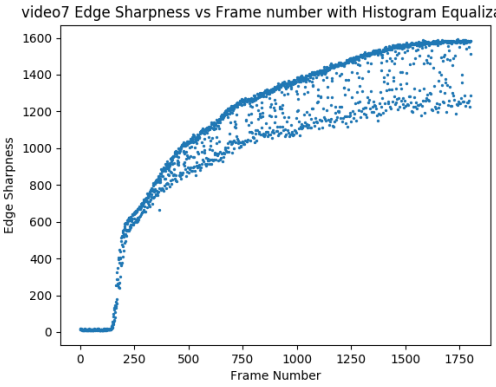

video5 Edge Sharpness vs Frame number without Histogram Equalization
Edge Sharpness
Frame Number


video5 Edge Sharpness vs Frame number with Histogram Equalization
Edge Sharpness
Frame Number


video6 Edge Sharpness vs Frame number without Histogram Equalization
Edge Sharpness
Frame Number


video6 Edge Sharpness vs Frame number with Histogram Equalization
Edge Sharpness
Frame Number


video7 Edge Sharpness vs Frame number without Histogram Equalization
Edge Sharpness
Frame Number


video7 Edge Sharpness vs Frame number with Histogram Equalization
Edge Sharpness
Frame Number

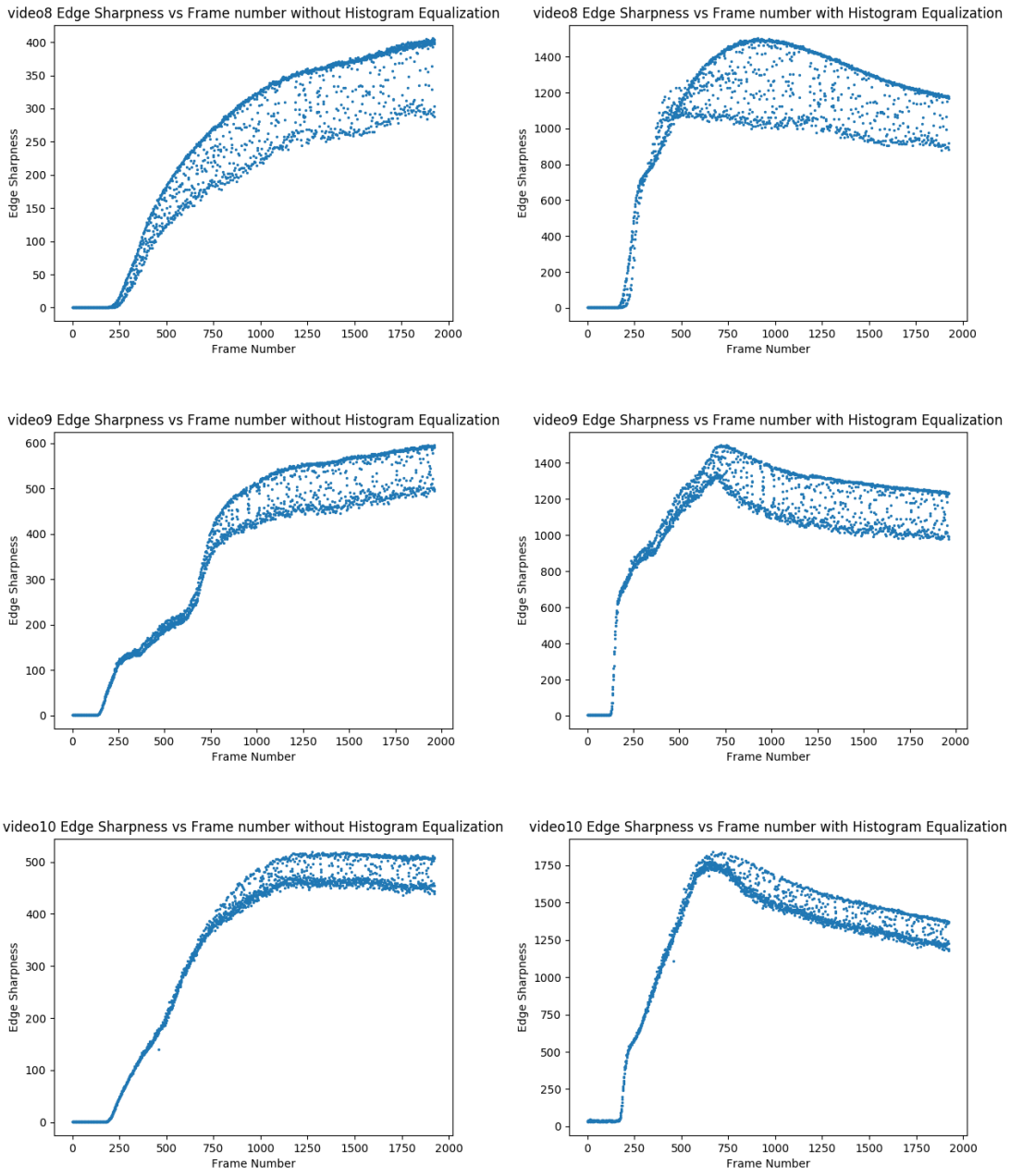


*Figure 36: (left) video 1 to 10 Edge Sharpness vs Frame number graphs **without** applying Histogram equalization, (right) video 1 to 10 Edge Sharpness vs Frame number graphs **after** applying Histogram equalization*

According to figure 36 (left), all the graphs show the same trend which is larger the frame number or longer the time, the edge sharpness increases as the brightness of the frame or the blood vessel increases. Most graph's gradient reduces over time except video 1 and 6. All the graphs have a common defect that the edge sharpness value fluctuated between a higher and lower value. As proven by the data in table 1, the frames with higher edge sharpness value should be the sharp frame and the frames with lower edge value should be the blurry frames. This defect in all the graphs provided the key element to separate of sharp and blur frames. Therefore, by looping along all the data points in the graph and finding the mean of certain number of current data point's neighboring data points, if the current data point is above the mean, it will be classified as sharp frame. If it is below the mean, it will classify as blurry frames.

As all the videos have a different brightness throughout the time, from very dark to bright which causes all the graphs to come with a positive gradient, it may affect the blur and sharp classification. Therefore, histogram equalization was experimentally applied to try to flatten up the graphs.

Histogram equalization (HE) is a method that improves the contrast in an image [36]. This method normally increases the global contrast of images when the useful data of the image is represented by similar contrast values. The intensities can be better distributed on the histogram by this adjustment. This allows areas with lower local contrast to obtain a higher contrast. Histogram equalization accomplishes this by spreading out the most frequent intensity values effectively which is shown by Figure 37.

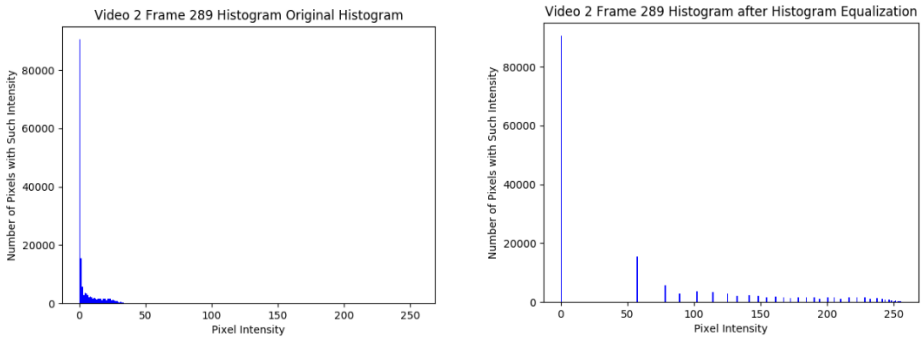


Figure 37:(left) video2 frame289 orginal grayscale histogram,
(right) video2 frame 289 grayscale histogram after histogram equalization

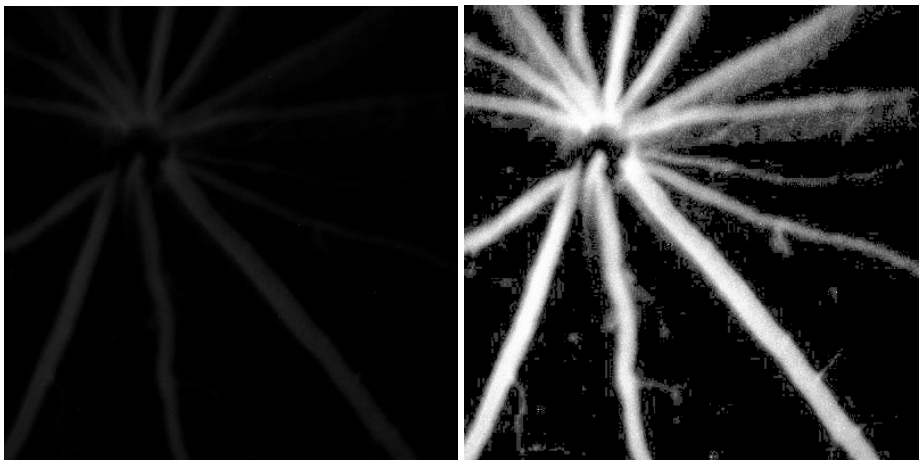


Figure 38: (left) orginal cropped grayscale video2 frame 289, (right) frame 289 after histogram equalization

Figure 38 shows that the image after histogram equalization shows a much better brightness. Histogram equalization is hypothesized to make the edge sharpness values influence less by the increasing brightness. This can theoretically flatten the graph and make the blur sharp classification more accurate.

Figure 35 (right) shows all the edge sharpness value against frame number graphs after applying histogram equalized to all frames. The fluctuation of edge sharpness value still exist which could be used to separate out blurry frames. The graphs of video 1, 2, 3, 4, 6 and 8 showed an obviously flatten gradient compared to the graph without histogram equalization. How the histogram equalization could affect the accuracy of the blur frames removal needed further examination. Another difference is that, the edge sharpness for the graph with histogram equalization is much higher than the one without histogram equalization. This is because histogram equalization will spread out the intensity values in the histogram which increased the overall pixel intensity to enhance the contrast of the images. Higher pixel intensity leads to higher value from the Sobel operator which increased the edge sharpness values.

4.3 Blurry Frames Removal Algorithm

The next step is to separate the frame number into two categories, the sharp and the blur set, based on the edge sharpness value. As mentioned before, this is accomplished by looping along all the data points in the graph and find the mean over certain number of the neighboring data points (**meanBox**), if the current data point is above the mean, it will be classified as sharp frame. If it is below the mean, it will classify as blurry frames. This may remove the data point in the very low fluctuation area like the frames before 500. Only the areas of the graph with a large fluctuation in edge sharpness need to be classified, therefore the range of values for a specific number of neighboring data points (**PtPbox**) will be taken into consideration for the current data point. Only the data points with a range of values which is the gap between the sharp edge value and the blur edge value is larger than a certain value (**tolerance value**) need to be classified. Otherwise, the data point is just classified as sharp frame. This tolerance value needs to be universal for all the videos. Therefore, the average of all the edge sharpness value of each image was taken to multiply with a value which is the tolerance value. This value controlled how large the range of the neighboring data points was needed for blur classification. Further experimentations were needed to find the optimal value for this tolerance value. This is performed by the following code:

```
tolerance_of_blur_v2_value = int(np.average(alldata[0:len(alldata), 1])*0.1)

for i in range(100, len(alldata)):

    #the mean value for a certain size of data range around the current data point
    mean = np.mean(alldata[i-meanBox:i+meanBox, 1])

    # if the current data point is larger than the mean of the certain size of data range
    # around the current data point, it is sharp. if smaller than the mean, it is blurry
    if np.ptp(alldata[i-ptpBox:i+ptpBox, 1]) > tolerance_of_blur_v2_value:
        if alldata[i][1]>= mean:
                sharpdata.append(alldata[i])

        else:
                blurdata.append(alldata[i])

    else:
        sharpdata.append(alldata[i])
```

There are 4 values that must be set. The **tolerance value**, the size of **mean box**, size of the **PtPBox** and the **Sobel Kernel size**. Also, how the histogram equalization affects the accuracy on sharp blur classification needs to be experiment.

4.3.1 Blurry Frames Removal Ground Truth

Before experimenting on these parameters, a set of ground truth value is required to calculate the accuracy on sharp blur classification. A piece of code base on python and Tkinter was written for users to manually classify frames into sharp and blur categories.

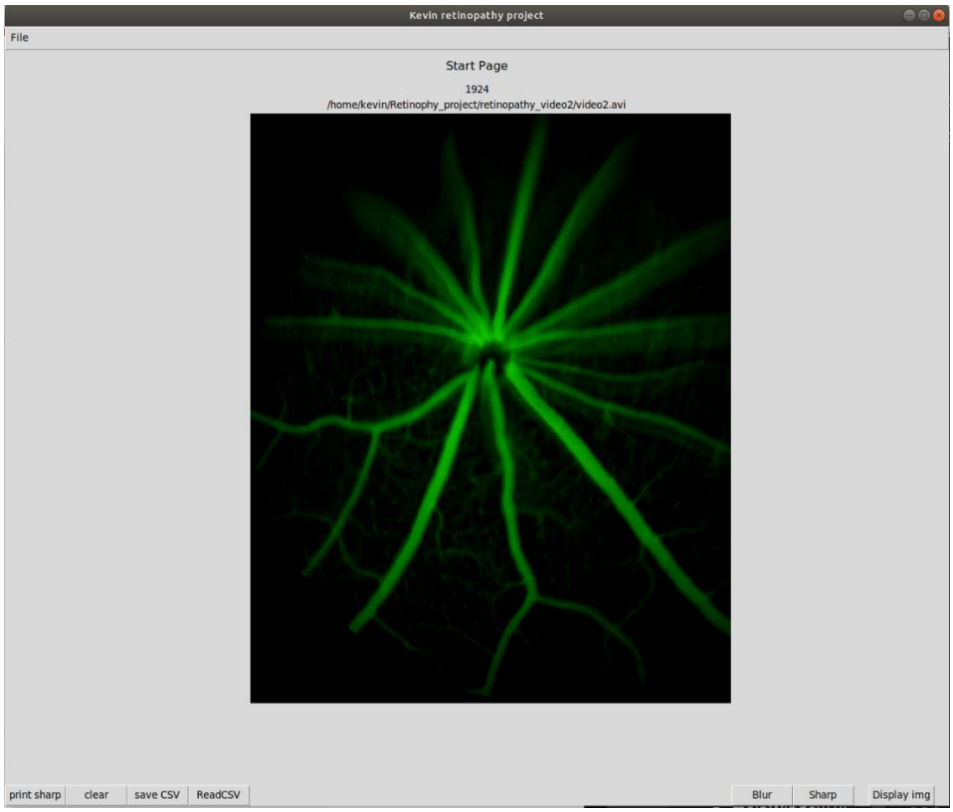


Figure 39: a pic of the gui for sharp blur classification to generate ground truth labels

This piece of code allow user to see every 2 frames of the video, and the user can click the blur button (q key in keyboard) or sharp button (w key in keyboard) to store the current frame number into the sharp list or the blur list. In the end, there will be two list of frame numbers, a sharp set and a blur list. This piece of code allows user to save these two lists into one CSV file. This manual frame classification was repeated three time to reduce the human errors. Therefore, there were three sharp and three blur list of frame-numbers. In the end, only the frame-numbers exist in all three of the sharp or blur list will be put in the final list. Finally, this generated one single set of ground truth sharp frame-numbers and blur frame-numbers.

When the frame numbers were separated into the sharp and the blur set based on the edge sharpness value, the number of frame numbers have in common with the ground truth sharp and blur frame number were counted and divide by the total number of ground truth sharp or blur frame to create a percentage for the sharp frames identification and blur frames identification accuracy, shorten as sharp accuracy and blur accuracy

This accuracy was just an indication but not the actual “accuracy” of the algorithm as the ground truth was only classified every 2 frames and some frames only had imperceptible blur which were very hard to distinguish by human eyes. Also, some frames were very dim and dark to classify. Therefore, the ground truth classification process was repeated three time to minimize this human error. After obtaining the ground truth frame numbers, the experiment for setting the parameters for the blurry frame removal is ready. Video 6 and 10 are excluded from this experiment, it is because the blurry frame in these two videos are very hard to distinguished by human eyes so no ground truth frame-number set could be generated.

4.3.2 PtPBox Training Experiment and Result Evaluation:

In this experiment, the MeanBox is set to 10 and the tolerance value was set to 0.1 to examine how different size PtP box affected the sharp and blur accuracy. PtP box size of 2, 4, 8 10, 20, 30, 40, 50 and 100 were tested. Sobel kernel size 3, 5 and 7 were also tested. Every frame was crop from 200 to 600 x and y pixel coordination.

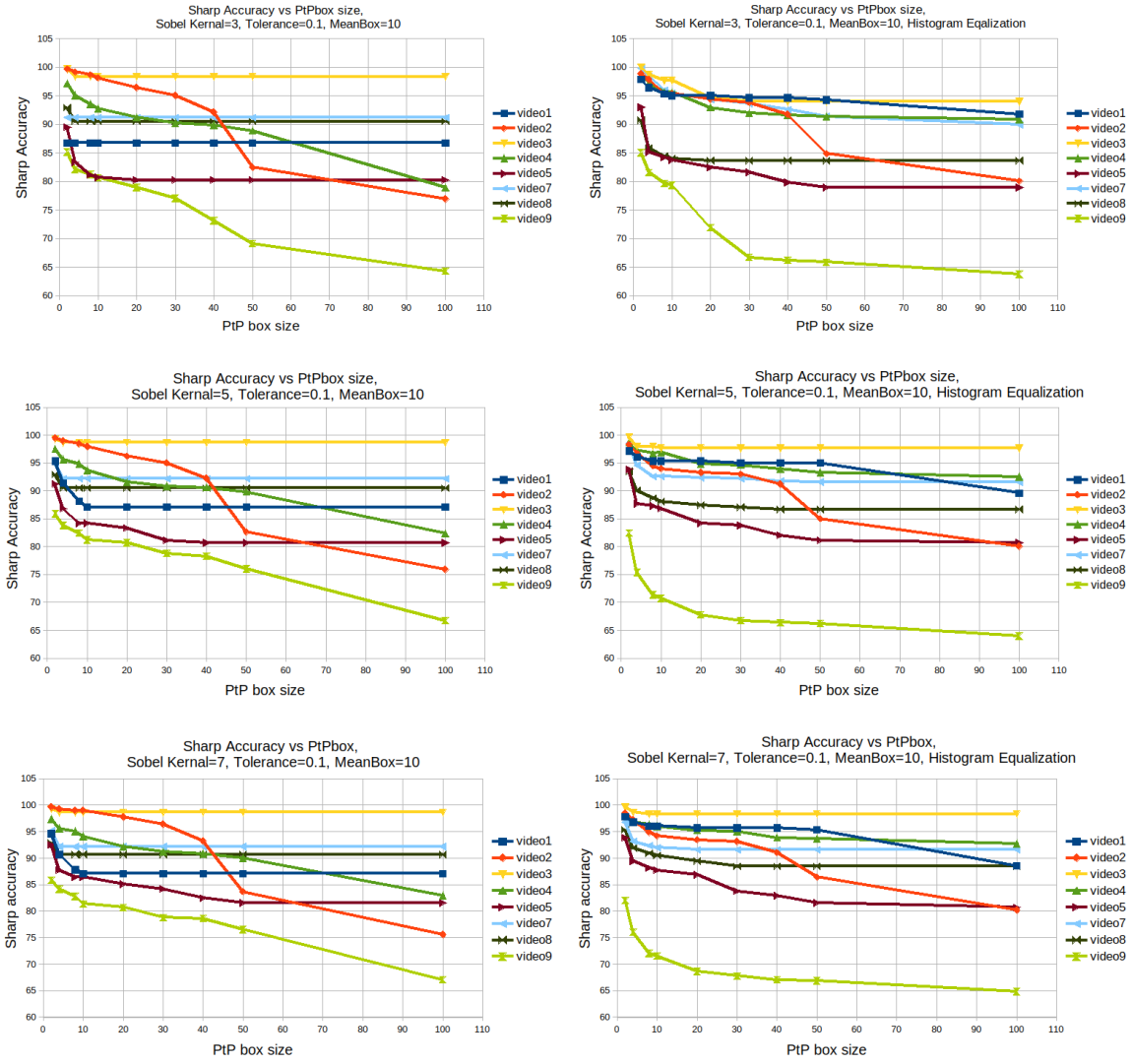


*Figure 40: (left) sharp accuracy vs PtPbox size for Sobel kernel size 3, 5 and 7, tolerance=0.1 and MeanBox=10, **without** applying histogram equalization, (right) sharp accuracy vs PtPbox size for Sobel kernel size 3, 5 and 7, tolerance=0.1 and MeanBox=10, **after** applying histogram equalization*

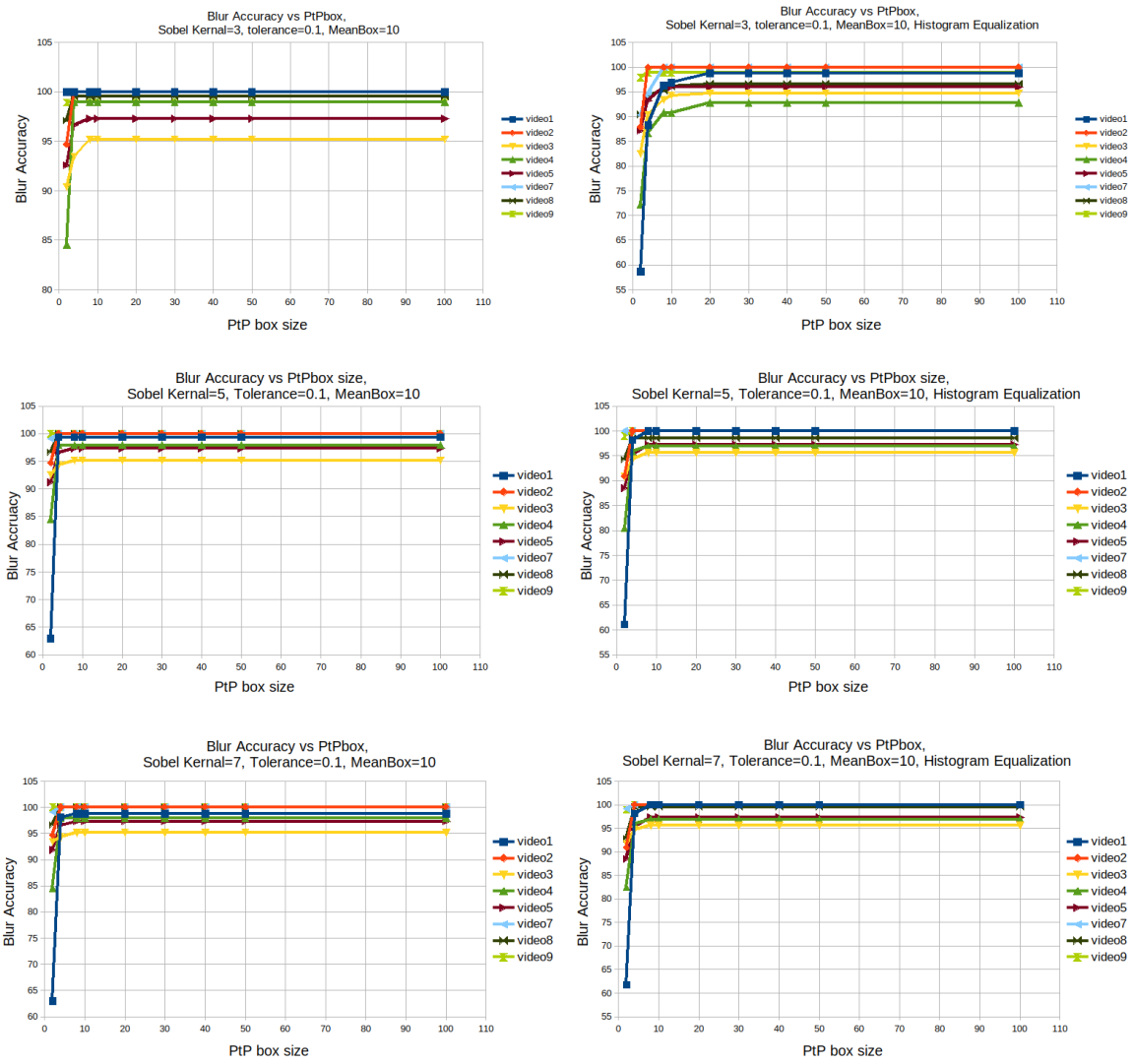


*Figure 41: (left) blur accuracy vs PtPbox size for Sobel kernel size 3, 5 and 7, tolerance=0.1 and MeanBox=10, **without** applying histogram equalization, (right) blur accuracy vs PtPbox size for Sobel kernel size 3, 5 and 7, tolerance=0.1 and MeanBox=10, **after** applying histogram equalization*

For all the results, the blur accuracy tended to be the lowest, but the sharp accuracy was the highest when the PtP box size was below 10. This indicated that a lot of the blurry frame were classified as sharp frames when the PtP box size was below 10.

The blur accuracy behavior was very similar between results with and without histogram equalization (**HE**) (Figure 41). The blur accuracy tended to have a sharp increase then remained at a constant value for all results when the PtPbox size was above 10, which means the blur accuracy was lower when the PtPbox size was below 10. The blur accuracy was not affected by the PtPbox size when it was above 10. This indicated that the PtPbox should be set 10 or above.

According figure 40, when the PtPbox size was set beyond 10 while no HE was applied, the sharp accuracies of videos 2, 4 and 9 were decreasing for all three Sobel kernel size, video 5 also showed sign of decrease when the Sobel kernel size was 5 and 7 (Figure 40 left). After applying HE with PtP box size set beyond 10, video 1, 2, 4, 5, 7 and 9 showed a decrease in sharp accuracy for all three Sobel kernel size, video 3 showed sign of sharp accuracy reduction when the kernel size was 3 and video 8 showed sign of sharp accuracy reduction when the kernel size was 5 and 7 (Figure 40 right). This showed that increasing PtP box size reduced the sharp accuracy of some videos which means more frames were classified as blurry frame when PtP box size increased. Therefore, the PtP box size shouldn't be too large beyond 10. The result also showed that HE made the sharp accuracy more sensitive to PtP box size as more video showed a decrease in sharp accuracy after applying HE.

In conclusion, when the PtP box size was set below 10, it resulted a high sharp accuracy but low blur accuracy. When the PtP box size was too much above 10, it resulted a sharp accuracy reduction, but the blur accuracy remained constant. Therefore, PtP box size 10 showed a very good balance between sharp and blur accuracy.

4.3.3 Tolerance Training Experiment and Result Evaluation:

In this experiment, the MeanBox was set to 10 and the PtPbox was set to 10 to examine how different tolerance value affected the sharp and blur accuracy. Tolerance value of 0, 0.01, 0.05, 0.1, 0.15 and 0.2 were tested. Sobel kernel size 3, 5 and 7 were also tested. Every frame was crop from 200 to 600 x and y pixel coordination.

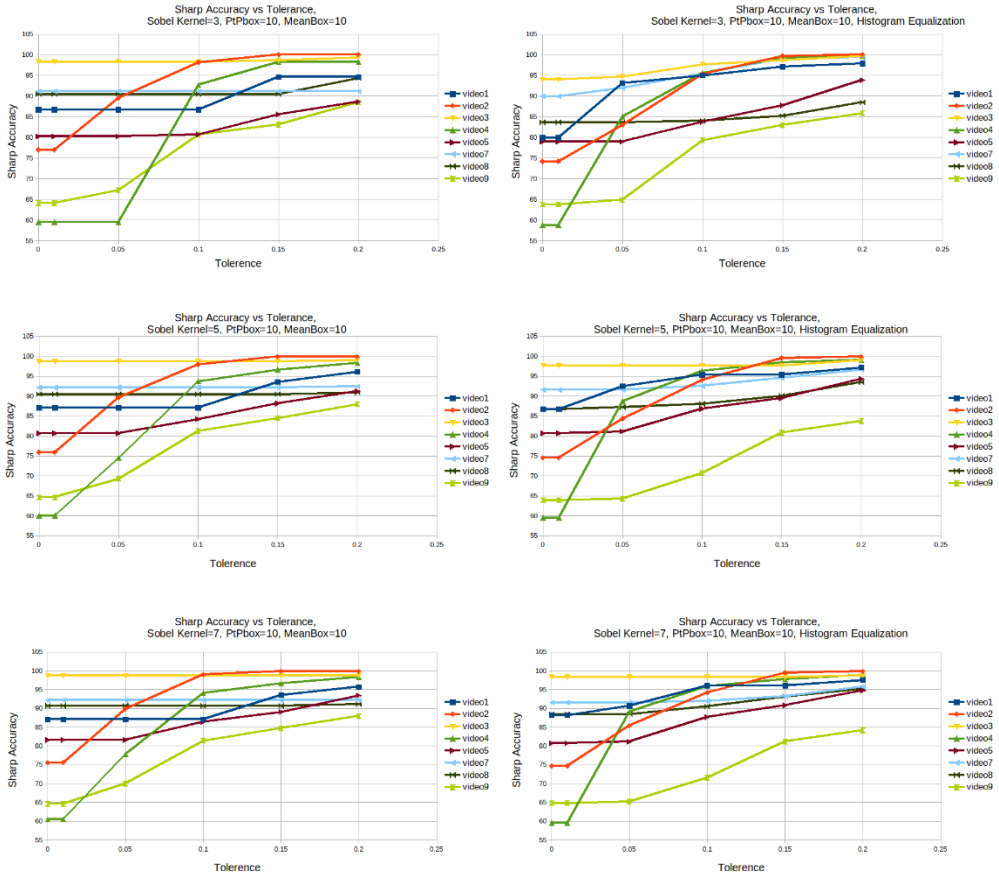


Figure 42: (left) sharp accuracy vs Tolerence value for Sobel kernel size 3, 5 and 7, PtPbox=10 and MeanBox=10, ***without*** *applying histogram equalization, (right) sharp accuracy vs Tolerence value for Sobel kernel size 3, 5 and 7, PtPbox=10 and MeanBox=10,* ***after*** *applying histogram equalization*

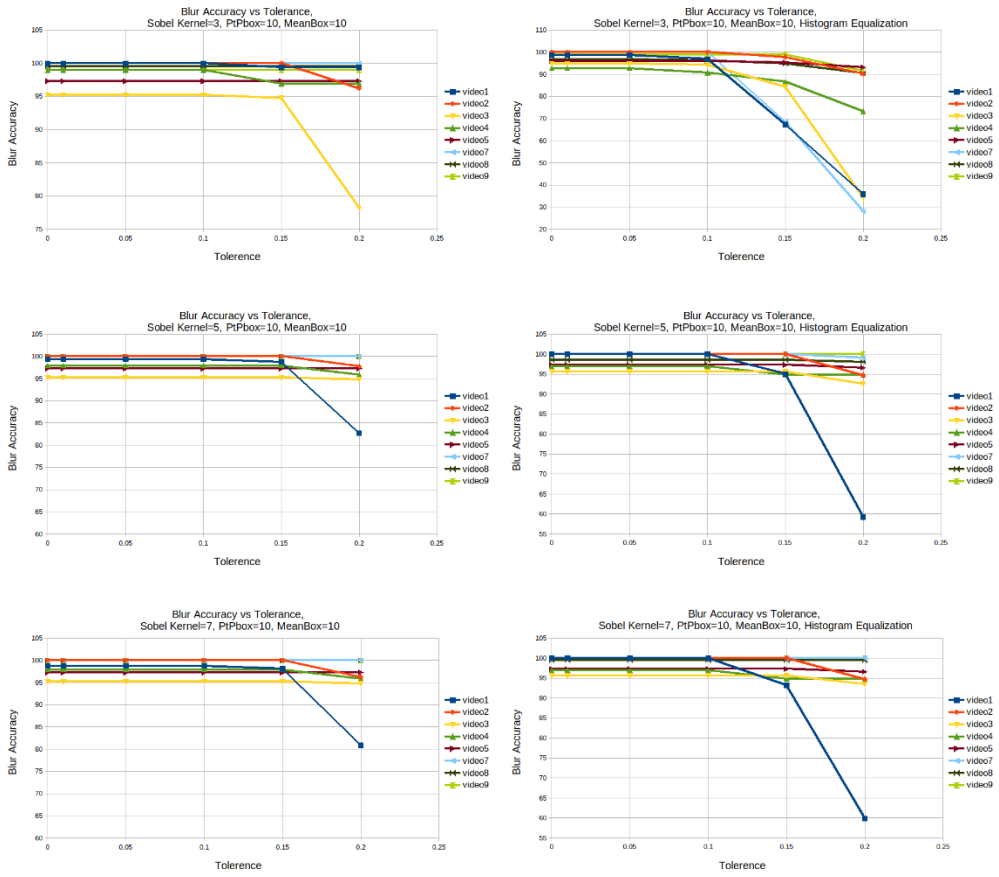


*Figure 43: (left) blur accuracy vs Tolerence value for Sobel kernel size 3, 5 and 7, PtPbox=10 and MeanBox=10, **without** applying histogram equalization, (right) blur accuracy vs Tolerence value for Sobel kernel size 3, 5 and 7, PtPbox=10 and MeanBox=10, **after** applying histogram equalization*

When no histogram equalization (HE) was applied, video 1, 2, 4, 5 and 9, (video 8 only for Sobel kernel 3) showed an increasing trend in sharp accuracy (Figure 42 left) when the tolerance value increase, but video 1, 2, 3 and 4 showed a decrease in blur accuracy when tolerance value increase above 0.1 for all Sobel kernel size (Figure 43 left).

After applying HE, all videos showed an increasing trend in sharp accuracy (Figure 42 right). According to Figure 43 (right), all videos showed a decrease in blur accuracy when Sobel kernel is 3 and the tolerance value is greater than 0.05, all videos except video 9 showed a decrease in blur accuracy when Sobel kernel was 5 and tolerance value greater than 0.1, only video 1, 2, 3, 4 and 5 showed a decrease when Sobel kernel was 7 and the tolerance value was greater than 0.1. The decrease in blur accuracy was much larger than the result without applying HE and this decrease in blur accuracy was most severe when the Sobel kernel was 3. Taking video 1 as the example, when the tolerance value was 0.2 and Sobel kernel was 3, applying HE made the blur accuracy dropped to 35% while the blur accuracy only dropped to 99% when no HE was applied. When the Sobel kernel was 5 and 7, applying HE caused blur accuracy to drop to 60% but the blur accuracy only dropped to 80% when no HE was applied. These results showed that HE made the blur accuracy more sensitive to tolerance value as more video showed more severe decrease in blur accuracy when tolerance value increased.

The result with and without HE showed the same trend that increasing tolerance value would increase sharp accuracy but decrease blur accuracy for some videos. This indicated that more frames were classified as sharp when the tolerance value was large, more frames were classified as blurry when the tolerance value was small. This also showed that the optimal value for the tolerance value was in the middle which could balance the sharp and blur accuracy to obtain the highest value for both accuracies. As most of the decrease in blur accuracy only happen when the tolerance value is larger than 0.1, which hinted that the tolerance value should not be set too much above 0.1 to maintain the highest blur accuracy. Although 0.1 tolerance value couldn't obtain the highest sharp accuracy, it is the best value to balance the highest sharp and blur accuracy.

4.3.4 Mean Box Training Experiment Result:

In this experiment, the tolerance value was set to 0.1 and the PtPbox was set to 10 to examine how different MeanBox size affected the sharp and blur accuracy. MeanBox size of 2, 4, 8, 10, 20, 40, 50 and 100 were tested. Sobel kernel size 3, 5 and 7 were also tested. Every frame was crop from 200 to 600 x and y pixel coordination.

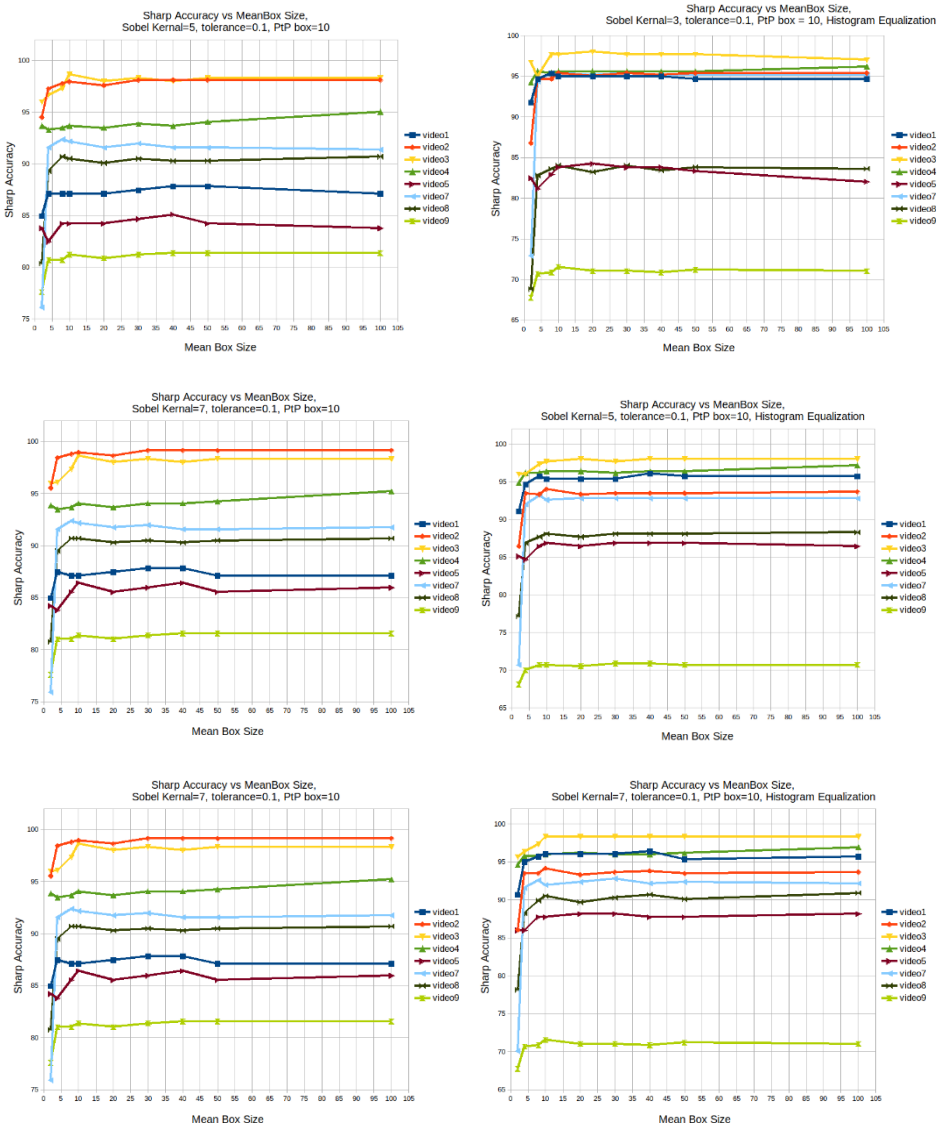


*Figure 44: (left) sharp accuracy vs MeanBox size for Sobel kernel size 3, 5 and 7, PtPbox=10 and Tolerence=0.1, **without** applying histogram equalization, (right) sharp accuracy vs MeanBox Size for Sobel kernel size 3, 5 and 7, PtPbox=10 and Tolerence=0.1, **after** applying histogram equalization*

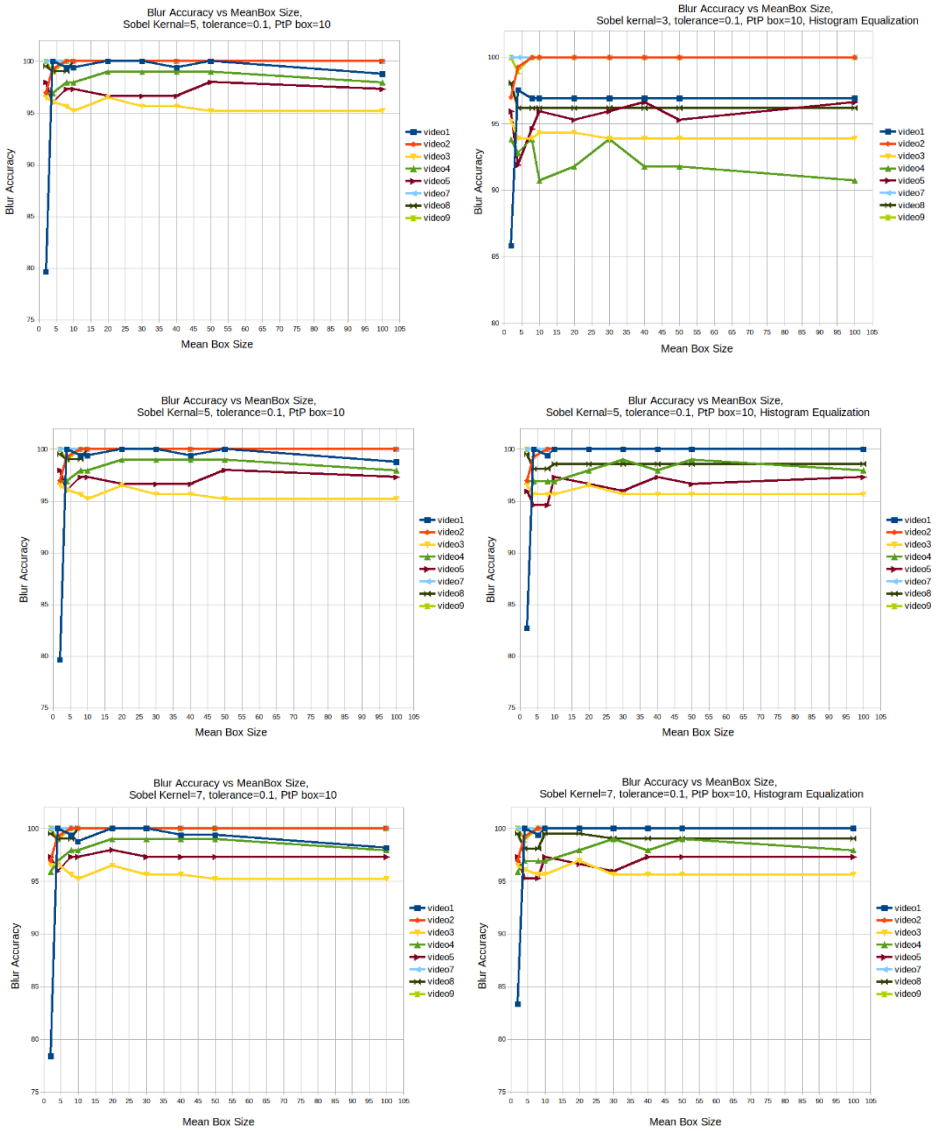


Figure 45: (left) blur accuracy vs MeanBox size for Sobel kernel size 3, 5 and 7, PtPbox=10 and Tolerence=0.1, ***without*** *applying histogram equalization, (right) blur accuracy vs MeanBox Size for Sobel kernel size 3, 5 and 7, PtPbox=10 and Tolerence=0.1,* ***after*** *applying histogram equalization*

For all the sharp accuracy results (Figure 44), the sharp accuracy showed a sharp increase when the mean box size was below 10 which means the sharp accuracy was lower when the mean box size was below 10. There was no obvious trend could be picked up when the mean box size was above 10, the sharp accuracy mildly fluctuated at the same level. This indicated that the mean box size shouldn't be set below 10.

For the blur accuracy result (Figure 45), a similar trend as the sharp accuracy was discovered for some video. When the mean box size was below 10, some video like 1, 2 and 4 showed a lower blur accuracy. A lot of the video showed fluctuation even the mean box size was above 10. The blur accuracy showed a more severe fluctuation after applying HE which made it very hard to distinguish the optimal value for highest blur accuracy (Figure 45 right). The first piece of hint was the mean box size had to be larger than 10 to obtain higher sharp and blur accuracy. Base on the blur accuracy result without the HE, mean box size 20 showed a high and roughly stable value for most videos.

The final decided value for the *Meanbox* size was *20*, the *PtPbox* size was *10* and the *tolerance value* was *0.1*.

4.3.5 Histogram Equalization and Sobel Kernel Size Experiment Result:

With the setting of mean box size 20, PtP box size 10 and tolerance value 0.1, the effect on different Sobel Kernel size and histogram equalization was further examined and the result is shown below.

	Without Histogram Equalization				*With* Histogram Equalization			
Video	Difference of **Sharp** Accuracy when **Kernel size increase from 3 to 5**	Difference of **Blur** Accuracy when **Kernel size increase from 3 to 5**	Difference of **Sharp** Accuracy when **Kernel size increase from 3 to 7**	Difference of **Blur** Accuracy when **Kernel size increase from 3 to 7**	Difference of **Sharp** Accuracy when **Kernel size increase from 3 to 5**	Difference of **Blur** Accuracy when **Kernel size increase from 3 to 5**	Difference of **Sharp** Accuracy when **Kernel size increase from 3 to 7**	Difference of **Blur** Accuracy when **Kernel size increase from 3 to 7**
1	None	None	+0.36%	None	+0.36%	+3.09%	+1.08%	+3.09%
2	None	None	+1.03%	None	-1.72%	None	-1.72%	None
3	+0.67%	None	+0.67%	None	None	+2.08%	+0.33%	+2.62%
4	+0.58%	None	+0.77%	None	+0.77%	+6.19%	+0.58%	+6.19%
5	+3.07%	None	+4.39%	+1.76%	+2.19%	+1.35%	+3.95%	+1.35%
7	+0.20%	None	+0.40%	None	-2.41%	None	-2.81%	None
8	+0.61%	None	+0.81%	None	+4.45%	+2.39%	+6.48%	+3.35%
9	None	None	+0.17%	None	-8.97%	None	-8.45%	None

Table 6: Show the difference in sharp and blur accuracy after increase Kernel size from 3 to 5 and frome 3 to 7

	Kernel Size = 3		Kernel Size = 5		Kernel size = 7	
Video	Difference of **Sharp** Accuracy **after Histogram Equalization**	Difference of **Blur** Accuracy **after Histogram Equalization**	Difference of **Sharp** Accuracy **after Histogram Equalization**	Difference of **Blur** Accuracy **after Histogram Equalization**	Difference of **Sharp** Accuracy **after Histogram Equalization**	Difference of **Blur** Accuracy **after Histogram Equalization**
1	+7.89	-3.09	+8.24%	None	+8.60%	None
2	-2.58%	None	-4.30%	None	-5.33%	None
3	+0.67%	-2.18%	None	None	+0.33%	+0.44%
4	+2.69%	-7.22%	+2.88%	-1.03%	+2.50	-1.031%
5	+3.07%	-1.35%	+2.19%	None	+2.63%	-1.35%
7	+3.81%	None	+1.20%	None	+0.60%	None
8	-6.28%	-3.83%	-2.43%	-1.44%	-0.6%	-0.4785
9	-1.14%	None	-10.34%	None	-10%	None

Table 7: showed the difference in sharp and blur accuracy after applying histogram equalization of all three kernel sizes.

Table 6 showed the difference in sharp and blur accuracy when the Sobel kernel size increase from 3 to 5 and from 3 to 7 for both the results with and without HE. For the data without applying HE, the blur accuracy is not very sensitive to the Sobel kernel size. Most videos except video 5 showed no changed in blur accuracy when the kernel size increase showed an increase in blur accuracy at kernel size 7. When the kernel size increased from 3 to 5, video 3 to 8 showed increase in sharp accuracy. When kernel size increased from 3 to 7, all video showed increase in sharp accuracy. This suggested that a larger kernel size could increase both the sharp and blur accuracy when no histogram equalization was applied. For the data after applying HE, the blur accuracy was more sensitive to kernel size as more videos (1, 3, 4, 5 and 8) showed increase in blur accuracy compare to the data without HE. Video 1, 4, 5 and 8 showed increased in both sharp and blur accuracy when kernel size increased from 3 to 5 and 3 to 7. Video 2, 7 and 9 showed decrease only in sharp accuracy when the kernel size increased from 3 to 5 and 3 to 7 and with no changes in the blur accuracy. This showed that increasing kernel size when applying histogram equalization may reduce the sharp accuracy in some videos.

Table 7 showed the difference in sharp and blur accuracy after applying histogram equalization of all three kernel sizes. Video 2 and 9 showed decrease in sharp accuracy and video 8 showed decrease in both sharp and blur accuracy for all three kernel sizes. Video 4 showed increment in sharp accuracy but reduction in blur accuracy for all three kernel sizes. Video 1, 3 and 5 showed increment in sharp accuracy but reduction in blur accuracy when the kernel size was 3. Only video 1 kernel size 5 and 7, video 7 all three kernel sizes, video 5 kernel 5 showed increment in sharp accuracy without reduction in blur accuracy. Also, only video 3 showed increment in both sharp and blur accuracy for kernel size 7. There is no kernel size that allow increment in both sharp and blur accuracy for all videos. No video showed increment in sharp accuracy without reduction in blur accuracy for kernel size 3, only video 1, 5 and 7 for kernel size 5 and video 1, 3 and 7 for kernel size 7 showed increment in sharp accuracy without reduction in blur accuracy. Only 3 videos out of 8 videos could successfully increase either one of the accuracies without scarifying the other accuracy.

According to table 7, majority of the video showed reduction in blur accuracy after applying histogram equalization and only 3 videos out of 8 videos could successfully increase either one of the accuracies without scarifying the other accuracy. As mentioned before during table 6 analysis, increasing kernel size when applying histogram equalization may reduce the sharp accuracy in some videos according. Conversely when no HE was applied, increasing kernel size may increase both the sharp and blur accuracy and no sign of drop in blur accuracy was shown. According to the tolerance training experiment, HE made the blur accuracy more sensitive to tolerance value as more video showed more severe decrease in blur accuracy when tolerance value increased. According to the MeanBox training experiment, the blur accuracy showed a more severe fluctuation after applying HE. These results suggested that no histogram equalization should be applied for default and largest kernel size 7 should be used for the Sobel edge detection.

Final Default Setting for Blurry Frames Removal				
Mean Box	PtP Box	Tolerance	Sobel Kernel Size	Histogram Equalization
20	10	0.1	7	None

Table 8: Shows the final default setting for blurry frames removal

video	Sharp Accuracy	Blur Accuracy
1	87.5%	100%
2	98.6%	100%
3	98.0%	96.5%
4	93.7%	99.0%
5	85.5%	98.0%
7	91.8%	100%
8	90.3%	100%
9	81.0%	100%

Table 9: Shows the sharp and blur accuracy with the final default setting

After all these experiments, the final setting for blur detection algorithm is 20 for the mean box size, 10 for the PtP box size, 0.1 for the tolerance value, 7 for the Sobel kernel size and no histogram equalization needed to apply. The resulting fluorescent intensity versus time graph after blurry frame removal is shown below. The sharp accuracy is above 80% and the blur accuracy is above 96%.

4.4 Final Result of Blurry Frames Removal

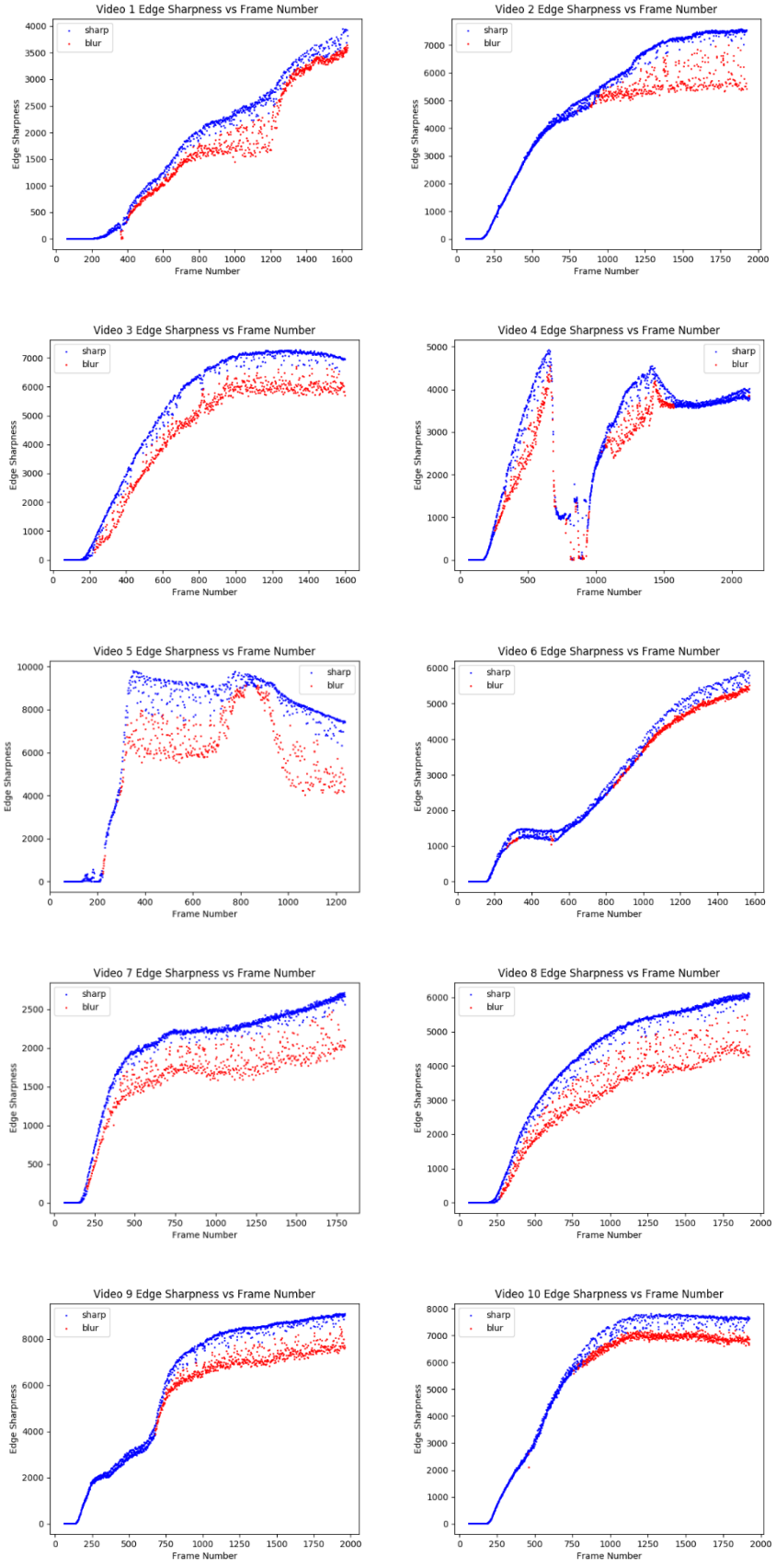


Figure 46: All 10 videos resulting Edge Sharpness vs Frame number graph after blurry frame removal with default setting, red is blurry data points, blue is sharp data points

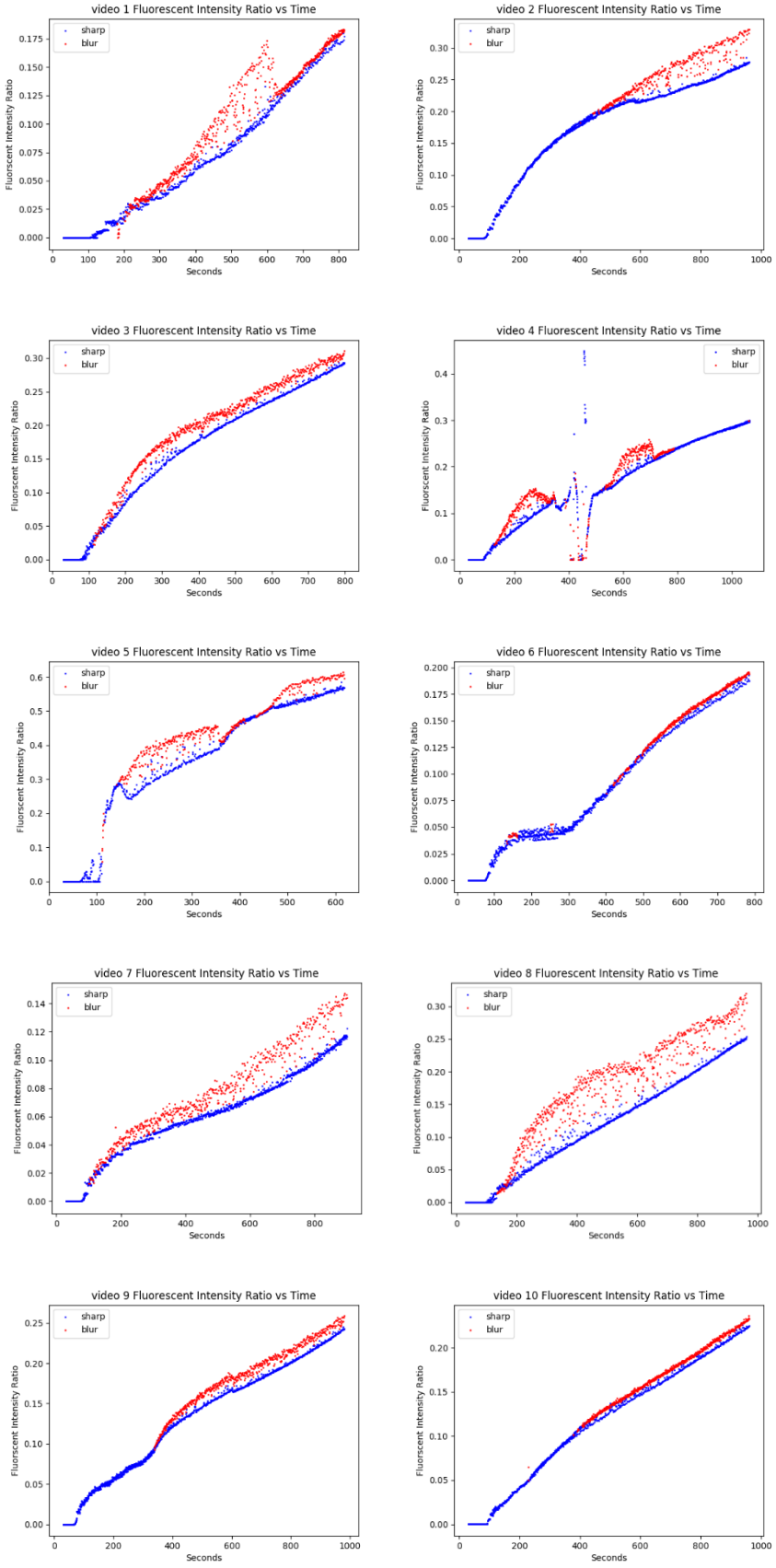


Figure 47: All 10 videos resulting Fluorescent Intensity Ratio vs time graph after blurry frame removal with default setting, red is blurry data points, blue is sharp data points

As mentioned in sections 3.7 and 4.2, blurry frames had lower edge sharpness value but higher fluorescent intensity ratio than the sharp image. The result of this blurry frame removal algorithm showed very good and correct result. Firstly, the red dots are the blurry data points and the blue dots are the sharp data points. For all 10 videos, the red data points had the lower edge sharpness and higher fluorescent intensity ratio. This supported the research results before and can therefore conclude that this blurry frame removal algorithm successfully selected out and eliminated the faulty fluorescent intensity ratio caused by blurry video frames.

Chapter 5: Gradient Extraction of the Fluorescent Intensity Ratio Graph:

5.1 Background of Gradient Extraction

The next step is to extract the gradient which is the Solute flux from the fluorescent intensity ratio vs time graph. Not the whole graph would be used to extract the gradient. Base on a discussion with Gamez [2], the decision on which section of the fluorescent intensity ratio vs time graph to pick for the gradient extraction was based on the exchange vessel fluorescent intensity vs time graph and the large vessel fluorescent intensity vs time graph.

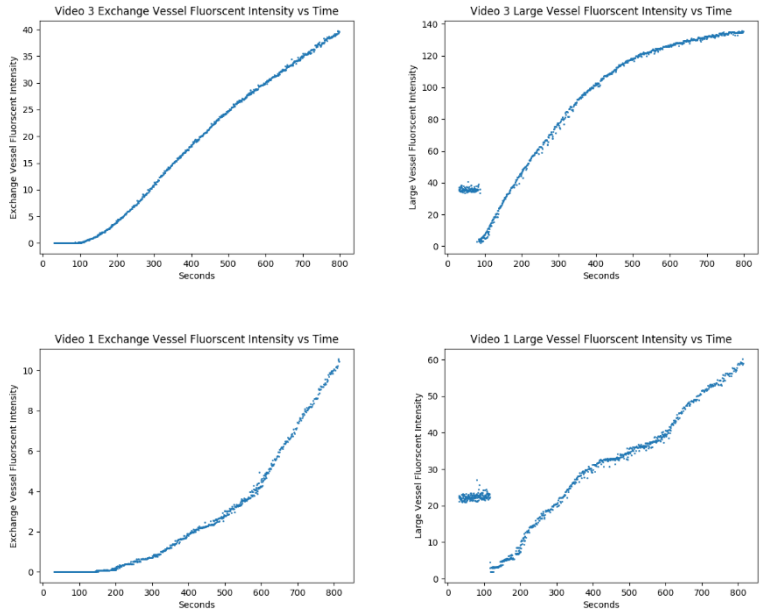


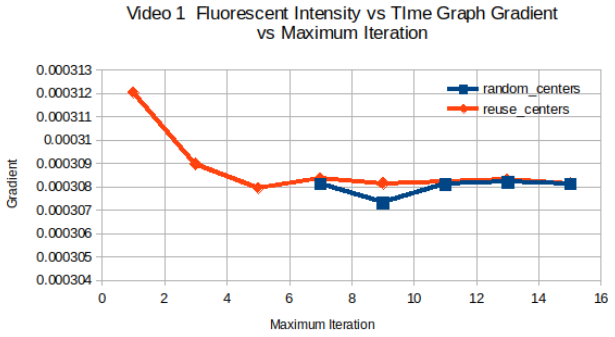
Figure 48: (top left)video 3 exchange vessel fluorescent intensity vs time graph (top right)video 3 large vessel fluorscent intensity vs time graph (bottom left) video 1 exchange vessel fluorscent intensity vs time graph (bottom right) video 1 large vessel fluorscent intensity vs time graph

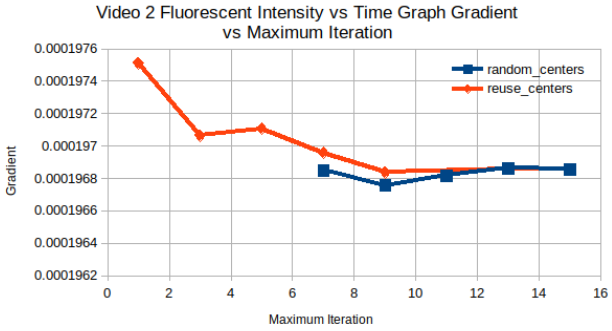
Gamez [2] mentioned the way to pick the time interval for gradient extraction was finding the region that is flattest in the large vessel fluorescent intensity vs time graph (Figure 48 right). The flattest part meant the large vessel is saturated with the sodium fluorescein which mean the large vessel reached the brightest point. In some cases, the large vessel may not be saturated so the large vessel fluorescent intensity vs time graph may not show a decline in gradient (). Therefore, the time interval for gradient extraction must be selected by the user. The exchange vessel fluorescent intensity vs time graph and the large vessel fluorescent intensity vs time graph need to be provided to the users for time interval selection.

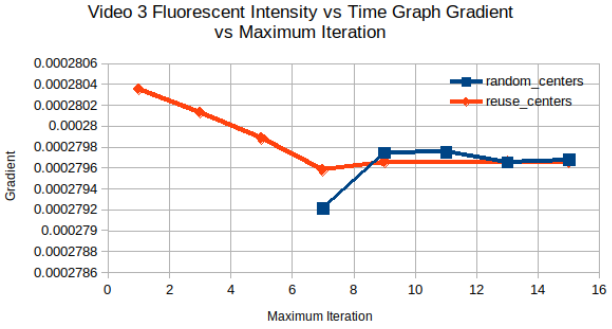
To obtain the gradient from the fluorescent intensity ratio vs time graph, linear regression needed to be performed. Linear regression is normally done by ordinary least squares linear regression. The gradient predicted by this method will easily be affected by outliners. The data points that diverge in a big way from the overall pattern are called an outliner. In order to improve the accuracy on the gradient, Random Sample Consensus (RANSAC) algorithm was then selected to use. RANSAC is a general parameter estimation approach designed to tackle large number of outliers in the data [41]. This will keep the gradient prediction free from the effect of outliners.

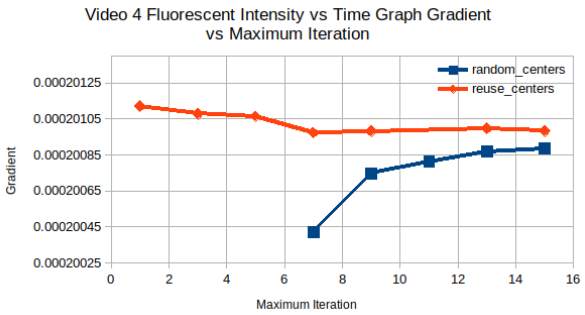
5.2 Difference in Gradient between Random Initial Centers K-means clustering and Reuse Centers K-means clustering:

Gamez [2] had provided all the gradient she found and the time interval she used previously (appendix). The same time intervals were used to compare the gradient obtain by K-means clustering with Sobel edge detection and Gamez's manual data extraction method, and how different maximum iteration affect the value of the gradient. The setting for blur detection algorithm was 20 for the mean box size, 10 for the PtP box size, 0.1 for the tolerance value, 7 for the Sobel kernel size and no histogram equalization applied. Epsilon 0.1 for the K-means clustering. Every frame was crop from 200 to 600 x and y pixel coordination. Maximum iteration 7, 9, 11, 13 and 15 were tested with random initial centers K-means clustering (blue lines). Maximum iteration 1, 3, 5, 7, 9, 13 and 15 were tested with reuse center K-means clustering (red lines).

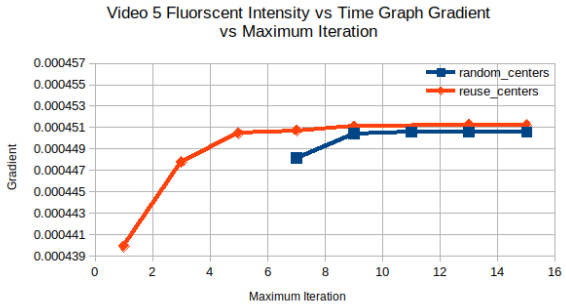




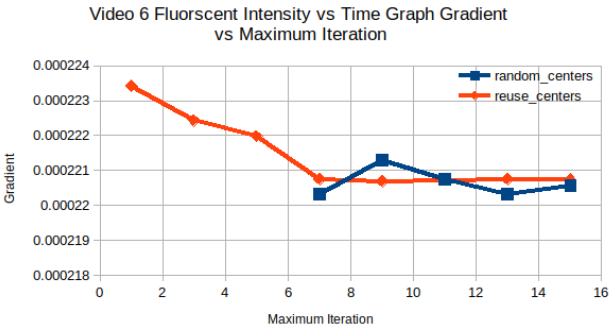



Video 4 Fluorescent Intensity vs Time Graph Gradient vs Maximum Iteration
random_centers
reuse_centers
Gradient
Maximum Iteration

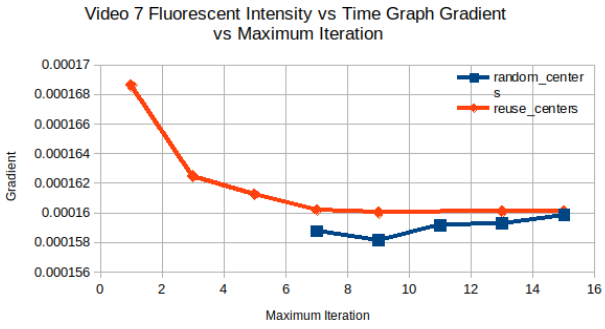

Video4 Time To Run K-means Clustering vs Maximum Iteration
original
reuse_centers
Time(seconds)
Maximum iteration


Video 5 Fluorscent Intensity vs Time Graph Gradient vs Maximum Iteration
random_centers
reuse_centers
Gradient
Maximum Iteration

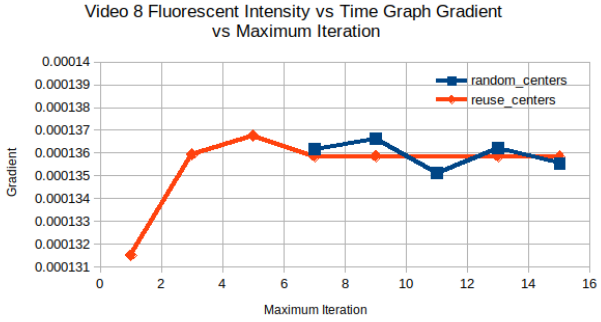

Video5 Time To Run K-means Clustering vs Maximum Iteration
original
reuse_centers
Time(seconds)
Maximum iteration


Video 6 Fluorscent Intensity vs Time Graph Gradient vs Maximum Iteration
random_centers
reuse_centers
Gradient
Maximum Iteration


Video6 Time To Run K-means Clustering vs Maximum Iteration
original
reuse_centers
Time(seconds)
Maximum iteration


Video 7 Fluorescent Intensity vs Time Graph Gradient vs Maximum Iteration
random_centers
reuse_centers
Gradient
Maximum Iteration


Video7 Time To Run K-means Clustering vs Maximum Iteration
original
reuse_centers
Time(seconds)
Maximum iteration


Video 8 Fluorescent Intensity vs Time Graph Gradient vs Maximum Iteration
random_centers
reuse_centers
Gradient
Maximum Iteration


Video8 Time To Run K-means Clustering vs Maximum Iteration
original
reuse_centers
Time(seconds)
Maximum iteration

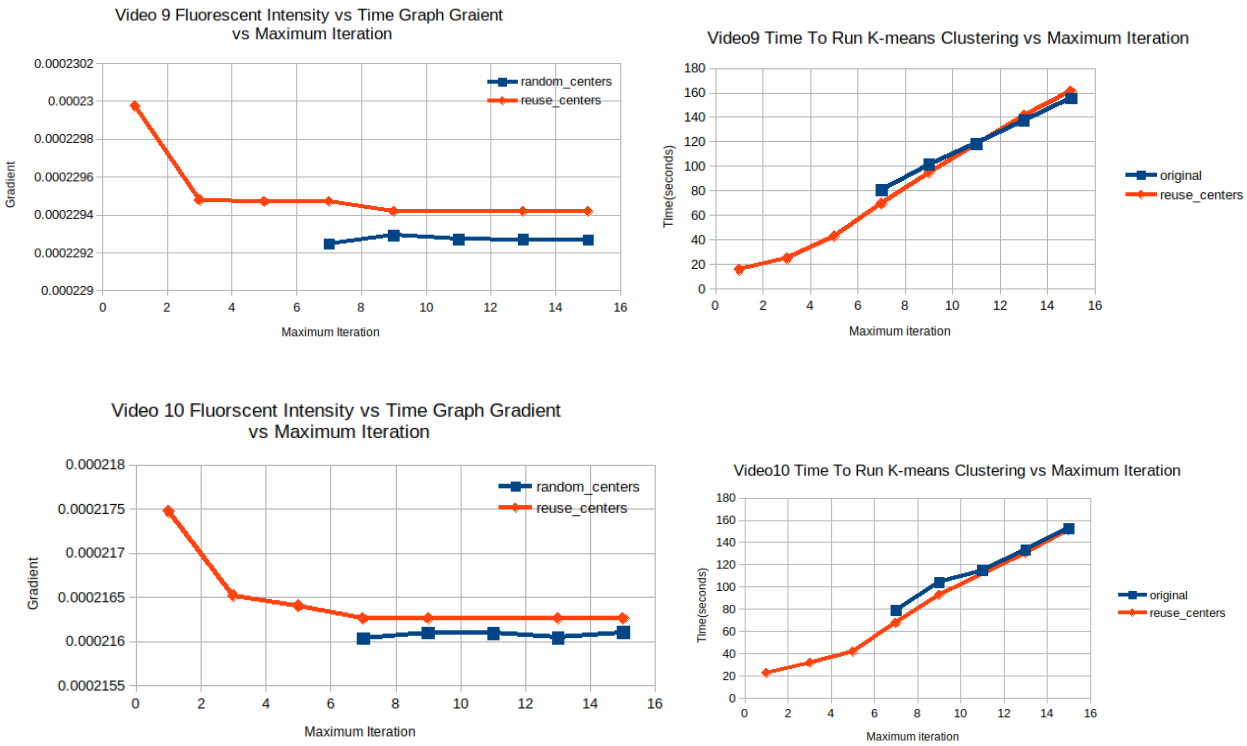


Figure 49: (left) all 10 videos' gradient value vs maximum iteration graph
*(right) all 10 videos' time to run K-means clustering vs maximum interation (**original means random initial centers**)*

According to all the gradient vs maximum iteration graphs, the gradients generated by reuse centers K-means clustering showed an obvious either decent or accent in gradient value when the maximum iteration was below 7, the gradient value tended to be stable when the maximum iteration was above 7. As mentioned in the section 3.6 and 3.8, no visible difference in the FIR vs time graphs with different maximum iteration but these data showed that imperceptible difference did exist and affect the gradient value. The gradients generated by reuse centers K-means clustering tended to be more stable than the gradients generated by the random initial centers K-means clustering when maximum iteration was above 7 and the difference between them are relatively small except video 4. This suggested that the default maximum iteration should be set to 7 and above. The downside is the runtime of reuse center K-means clustering with iteration 7 and above is the very similar to the runtime of random initial centers K-means clustering. The time advantage of reuse center K-means clustering will be scarified for the accuracy of the gradient value. As the gradient value difference between the two types of K-means clustering was very small and the maximum iterations between 1 to 6 are still usable with reuse center K-means clustering, reuse center K-means clustering will be implemented into the final software rather than the random initial centers K-means clustering. Maximum iteration 7 will be set as default to prioritize on the accuracy of the gradient value. The user can lower the maximum iteration to shorten the run time of the program.

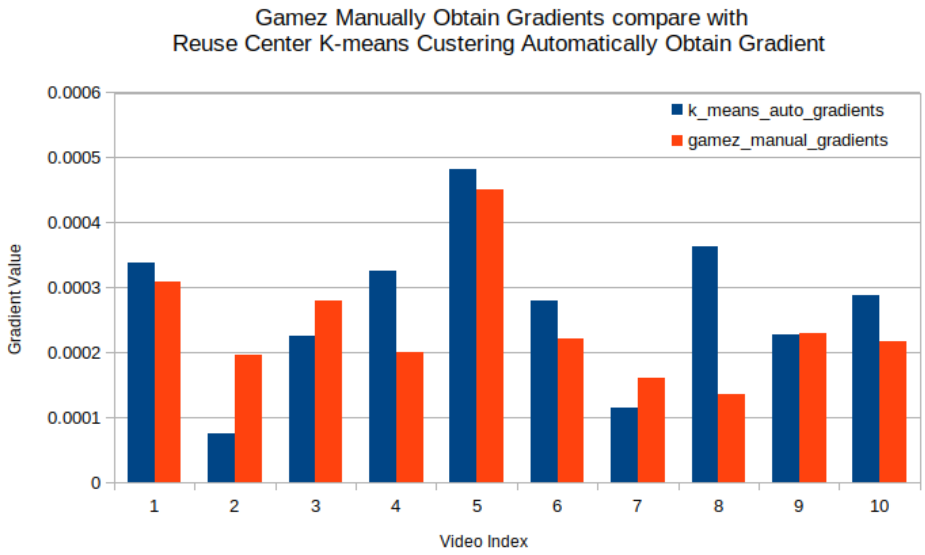


Figure 50: Histogram to show the difference in the gradient value automatically found by K-mean clutering with Sobel edge detection blurry frames removal and the gradient manually found by Gamez [2],
blue is K-means found by this project, orange is Gamez[2] gradient values

The gradient obtained from reuse centers K-means clustering with maximum iteration of 7 and Sobel edge detection blurry frames removal were compared with the gradient obtained manually by Gamez. Gamez gradients cannot be used as ground truth. Firstly, Gamez only used 20 data points for each of the gradient extraction, the code of this experiment used every frame of the video which has a huge advantage on sample size to improve accuracy. Gamez used two very small regions of interest on the capillaries to measure the fluorescent intensity ratio, and the two ROI had to be adjusted manually due to blurry frames caused by mice heartbeat, this involved a lot of human error as manually moving the ROI to a different pixel coordination may affect the fluorescent intensity ratio. On the other hand, the code of this experiment use only one much larger ROI (400 x 400 pixel) and it will always be fixed to the same location (200 to 600 pixel-coordination at X and Y), and all the blurry frames were eliminated, and no need to manually adjust the position of ROI so no human error. Most of the gradient values are very similar except video 2 and 8. This can be caused by the inaccurate data points collected by Gamez due to human errors. To some extent, the Gamez's resulting gradient values proofs that the gradient found by this experiment code found are correct. This experimental code was ready to combine with a graphical user interphase to create a piece of usable software.

Chapter 6: Software Development:

6.1 GUI design Objectives:

The features need to be in the GUI:

- File dialog to let user select the video to analyze
- Allow user to see specific frame of the video to select the region of interest
- Apply K-means clustering to obtain the fluorescent intensity for each frame and use Sobel edge detection to remove blurry frames
- Show both the large vessel fluorescent intensity vs time graph and the exchange vessel fluorescent intensity vs time graph to allow user to select the appropriate time interval for gradient extraction
- Use RANSAC to predict the gradient
- Show Fluorescent Intensity vs time and Edge sharpness vs time before and after blur removal and also the RANSAC prediction graph
- Offer a set of default optimal parameter setting for the K-mean clustering and the Sobel edge detection blur removal algorithm, and also allow user to change and set their desire parameters
- Allow user to save all the sharp, blur and gradient prediction data points as csv so that they can replot or manipulate the data in any other graph plotting software like Excel

6.2 GUI library selection:

Python provide a large variety of GUI programming library like Tkinter, PySide , PyQt, etc. Tkinter(Tk interface") was a very good starting point for prototyping a GUI. It is because Tkinter is the standard Python interface to the Tk GUI toolkit[38]. Both Tk and tkinter support most Unix platforms and Windows systems. (Tk itself is not part of Python, it is maintained at ActiveState.) The very first prototype with Tkinter is showen below. There are total of three windows, one is the main control window, one to select the region of interest of the video and one to display the resulting graphs.

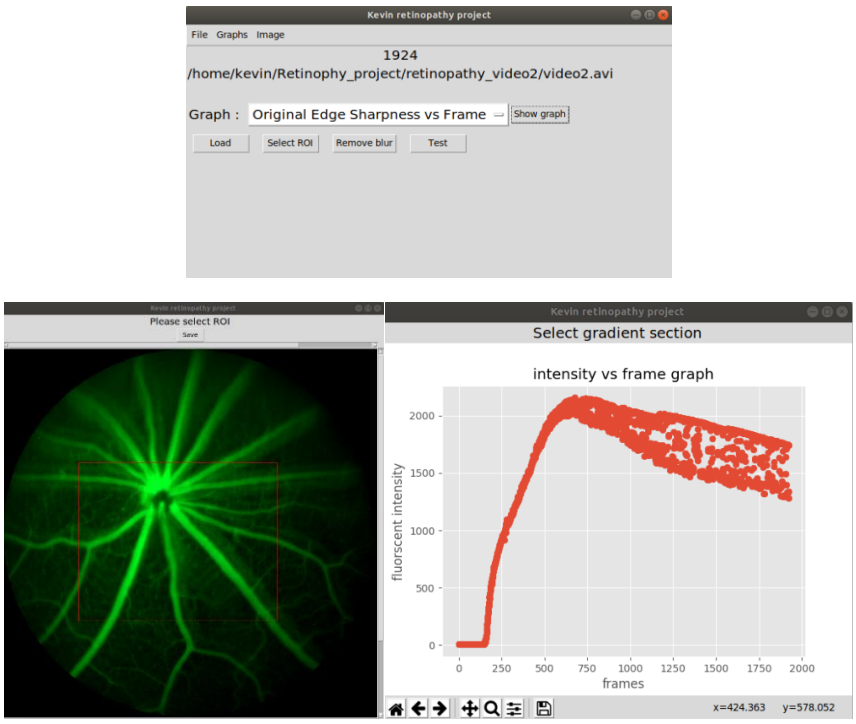


Figure 51: (top)the Tkinter prototype main window, (left) the Tkinter prototype to display a frame for user to use the rectangular selector to select ROI, (right) the Tkinter prototype one of the graph page, one graph per window only for the Tkinter prototype

The biggest obstacle when using Tkinter is Tkinter doesn't have a graphical UI designer software. All the positioning of the widget had to be coded manually. At this early stage, only a few widgets were implemented. At a more developed stage, the GUI will be more complicated as more functions needed to be implemented. Manually coding widgets positions and layout would be very time consuming and make it very difficult to create a sophisticated and appealing graphical user interphase (GUI). "tkinter is also famous for having an outdated look and feel, which has been vastly improved in Tk 8.5. Nevertheless, there are many other GUI libraries that you could be interested in." is quoted from the Python official documentation. The prototyped was built with Tk 8.6 and the design of the GUI was still very outdated and dull. A more robust GUI design tool was needed to make the GUI more appealing and offer a better user experience and provide graphical UI designer software to save time on widget positioning.

PyQt provided an easier and more robust environment to design GUI. PyQt is a set of Python 2 and 3 bindings for The Qt Company's Qt application framework and operates on all platforms supported by Qt including Windows, OS X, Linux, iOS and Android [39]. PyQt5 supports Qt v5. PyQt4 supports Qt v4 and will build against Qt v5. PyQt4 was selected for this project even though no new releases will be made for PyQt4. It is because PyQt4 existed a longer period of time than PyQt5, so it has a larger amount of documentation and tutorial to ease the learning curve a bit more. The difference between PyQt4 and PyQt5 is not very big so a future upgrade to PyQt5 for this software will not be too difficult.

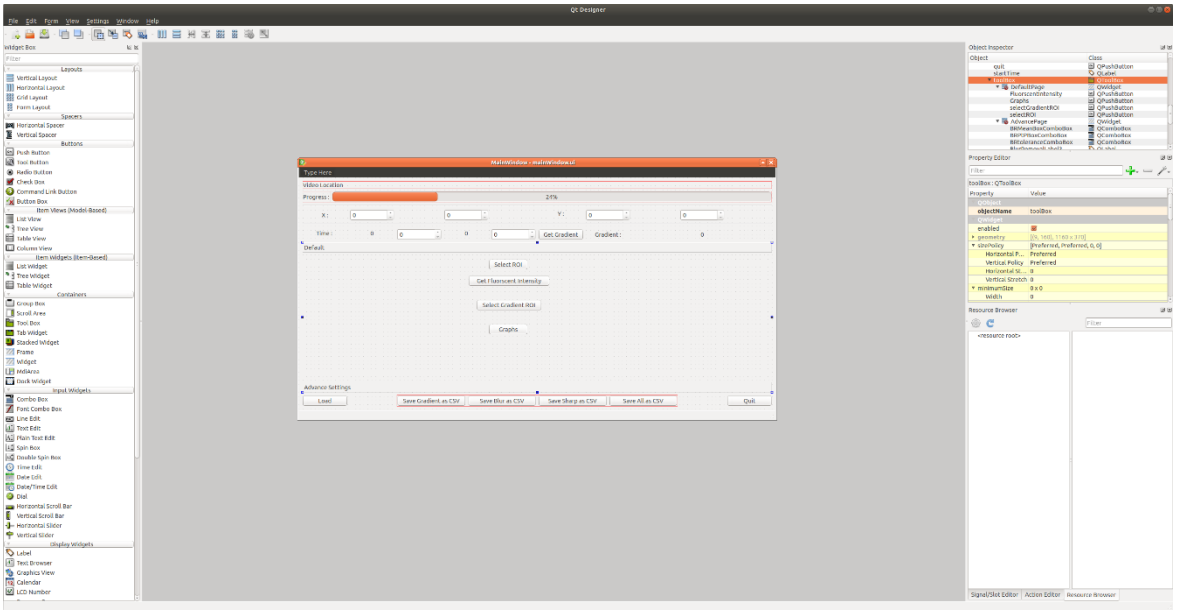


Figure 52: the Qt Designer window of designing the main window of this project software

PyQt4 biggest advantage over Tkinter is PyQt4 supports the usage of a graphical UI designer software called Qt Designer to generate .ui file that contains the layout and positioning of all the widgets. Qt Designer is the Qt tool for designing and building GUIs with Qt Widgets. This software allowed user to compose and customize the windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner and test them using different styles and resolutions. As the layout, positioning and the widgets declaration are included inside the separate .ui files, this saves a lot of lines of codes. It is because the PyQT4 python program just needed to load in the .ui file to use the designed window. The user only needed to code for connecting the Qt widgets to the corresponding functions to make the GUIs functional. This helps to save a lot of time and effort on GUIs designing. Also, Qt provide a lot of very useful pre-built widgets that Tkinter doesn't has, for example, a Qt toolbox which is a container that can show groups of items separated by tabs. This Qt toolbox was implemented to separate the default page and the advance setting of the software.

6.3 The GUIs structure:

There are total of 4 windows and two of the windows were designed with the Qt Designer, the main window (mainWIndow.ui) and the graph display page (graphPage.ui). The design of all the windows are shown below and the functions of each widgets are explained too.

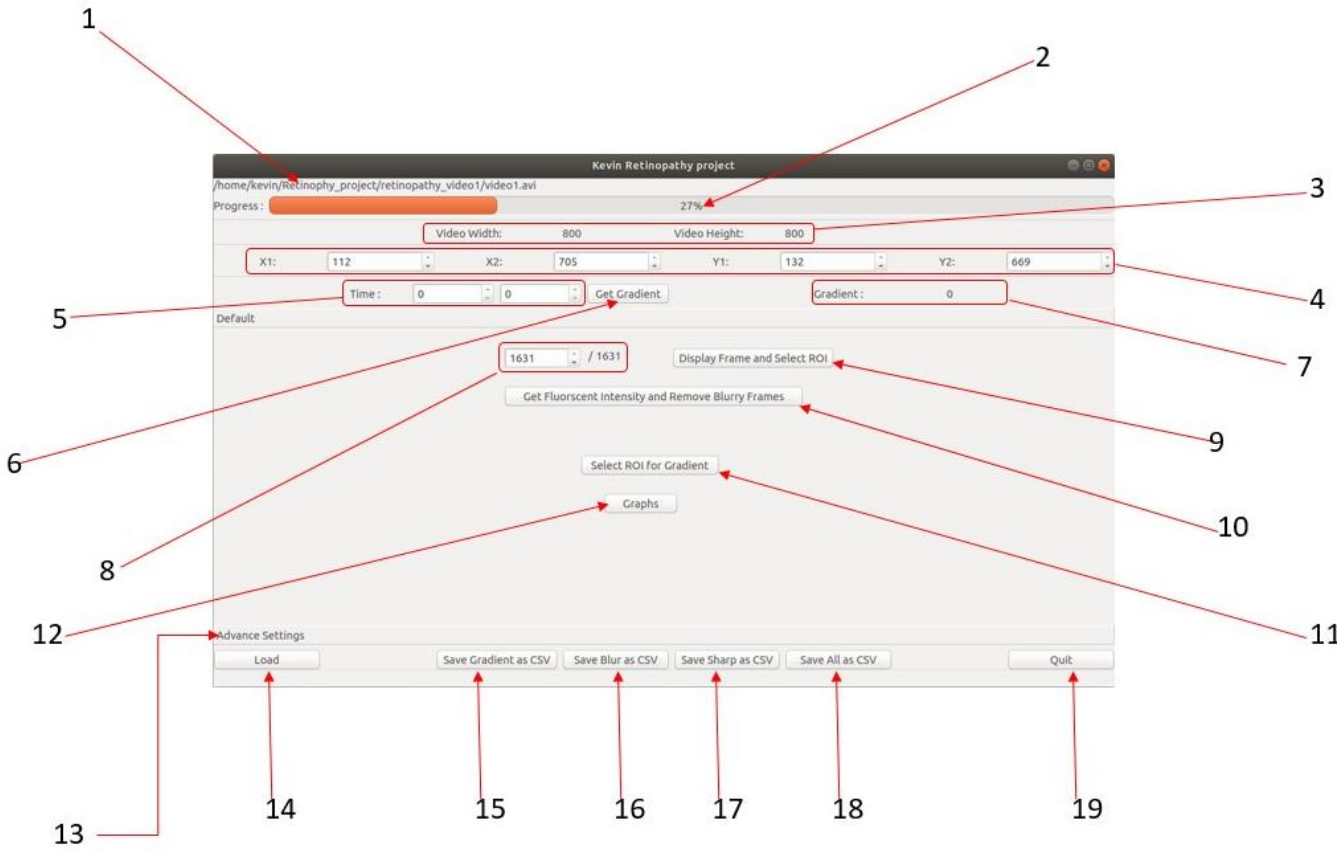


Figure 53: the main window with default setting tab

1. Show the file path of the user selected video file
2. The progress bar to show the progress of finding fluorescent intensity and removing blurring frame with K-means clustering and Sobel edge detection.
3. Display the width and height of the loaded video
4. Integer boxes (PyQT spinbox) to display or set the x and y pixel coordination of the users' region of interest of the video
5. Integer boxes to display or set the user's desire time interval for gradient extraction
6. Button to obtain the gradient
7. Label to display the gradient
8. Integer box to display or set the user's desire frame for region of interest selection
9. Button to show up the page to display the user's selected frame and a resizable rectangle selection for region of interest selection (Figure 55)
10. Button to start the process of finding fluorescent intensity and removing blurring frame with K-means clustering and Sobel edge detection
11. Button to activate the page for selecting time interval for gradient extraction (Figure 56)
12. Button to activate the graph page to see all the resulting graphs (Figure 57)
13. Button to activate the tab for advance setting page (Figure 54)
14. Button to start file Dialog window to allow user to select desire video file for analysis

15. Save the RANSAC gradient prediction data and the fluorescent intensity values within the user's selected time interval
16. Save all the **blur** data (Time(seconds), Fluorescent Intensity, Edge sharpness, Vein fluorescent Intensity, Exchange vessels fluorescent Intensity) as a CSV file
17. Save all the **sharp** data (Time (seconds), Fluorescent Intensity, Edge sharpness, Vein fluorescent Intensity, Exchange vessels fluorescent Intensity) as a CSV file
18. Save all the data without blur removal (Time (seconds), Fluorescent Intensity, Edge sharpness, Vein fluorescent Intensity, Exchange vessels fluorescent Intensity) as CSV file
19. Button to kill and close the program

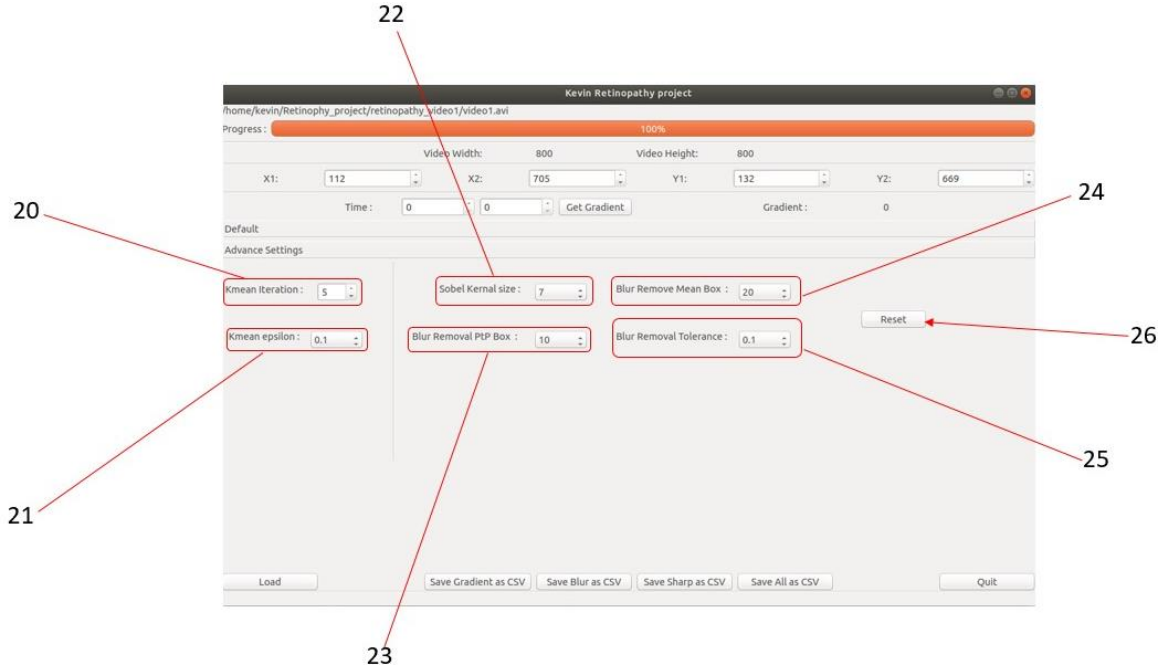


Figure 54: the main window with advance settings tab

20. Integer box to select different K-means iteration
21. Combo box to select different K-means epsilon
22. Combo box to select different Sobel kernel size
23. Combo box to select different blur removal PtP box size
24. Combo box to select different blur removal mean box size
25. Combo box to select different blur removal tolerance value
26. A button to reset all the advance settings to default value

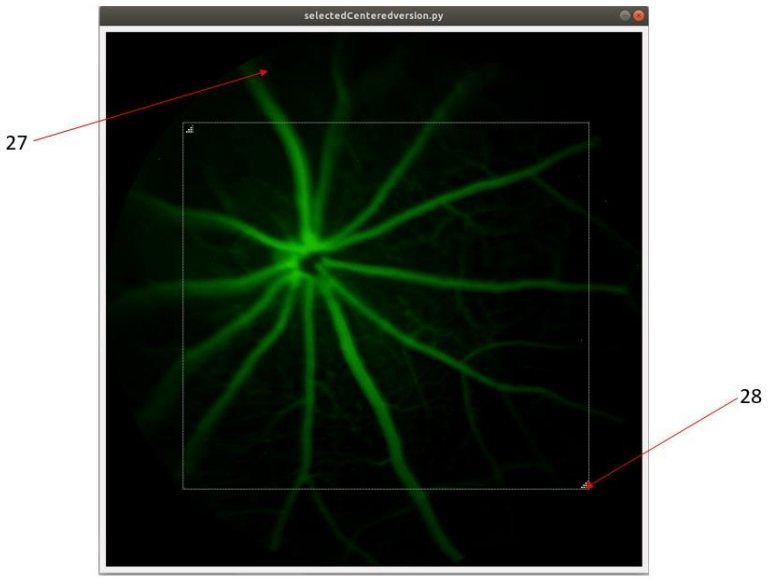


Figure 55: the pop-up window display the user's select frame with a resizable rectangular selector for user to select region of interest of the video

27. The display image of the user's selected frames
28. The resizable rectangular selector (right click to drag the selector, left click the corners to resize the selector)

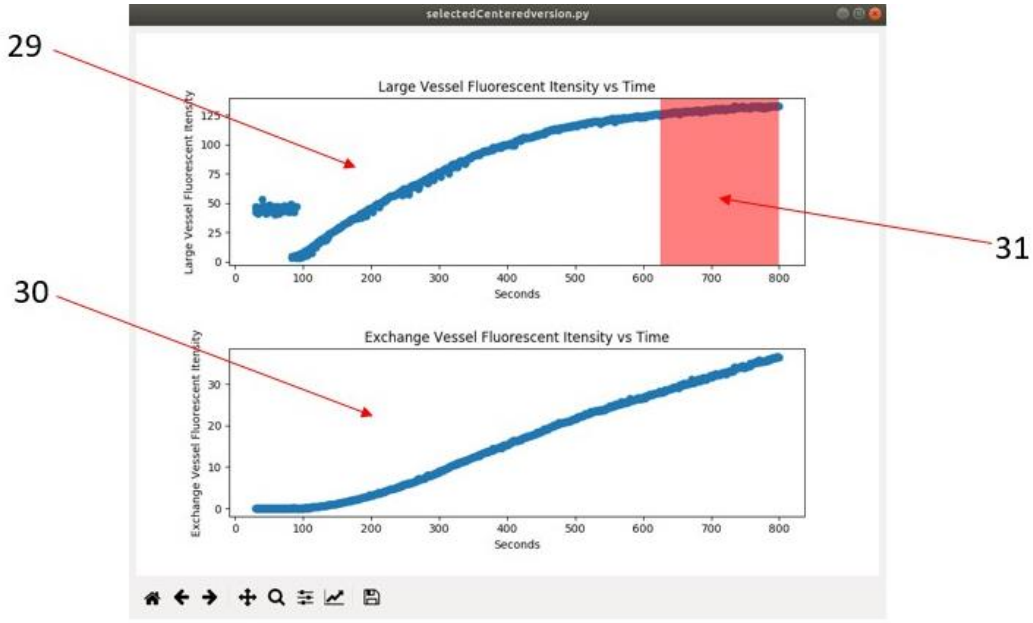


Figure 56: the pop-up window to show the Large vessel Fluorescent Intensity vs Time graph (top), and the Exchange vessel Fluorescent Intensity vs Time graph (bottom), a span selection only in the top graph to let user to select desire time interval for the gradient

29. The display of the large vessel fluorescent intensity vs time graph
30. The display of the exchange vessel fluorescent intensity vs time graph
31. The span selector for user to drag and select the desire time interval for gradient prediction

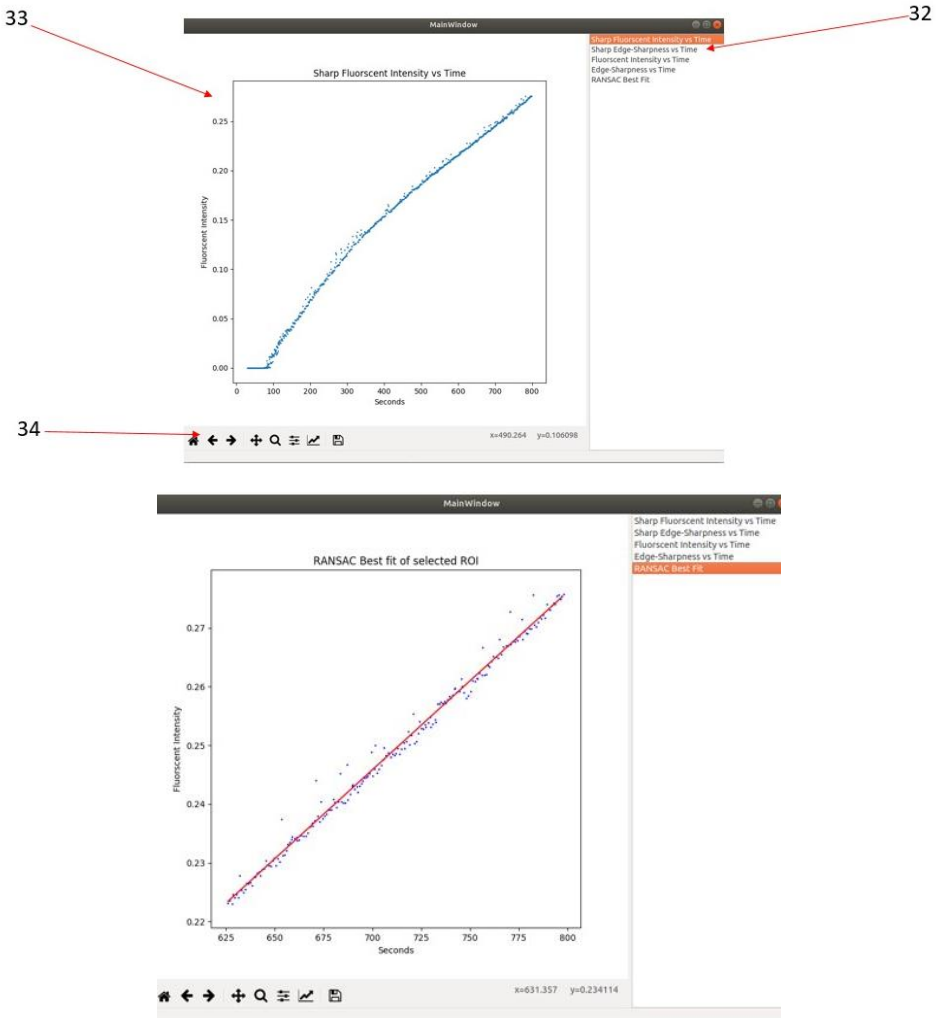


Figure 57: (top) the pop-up window show all the resulting graphs, the list on the right hand side allow user to click and switch to whatever graph they want to see, matplotlib tool bar at the bottom to let user to zoom, drag and save the graph, (bottom) shows the switch of graphs

32. The list of all the resulting graphs, the user clicks the name to display desire graphs
33. The display desired graph
34. The matplotLib tool bar to let the user to zoom in, drag or save the graph

6.4 Software Structures and Classes

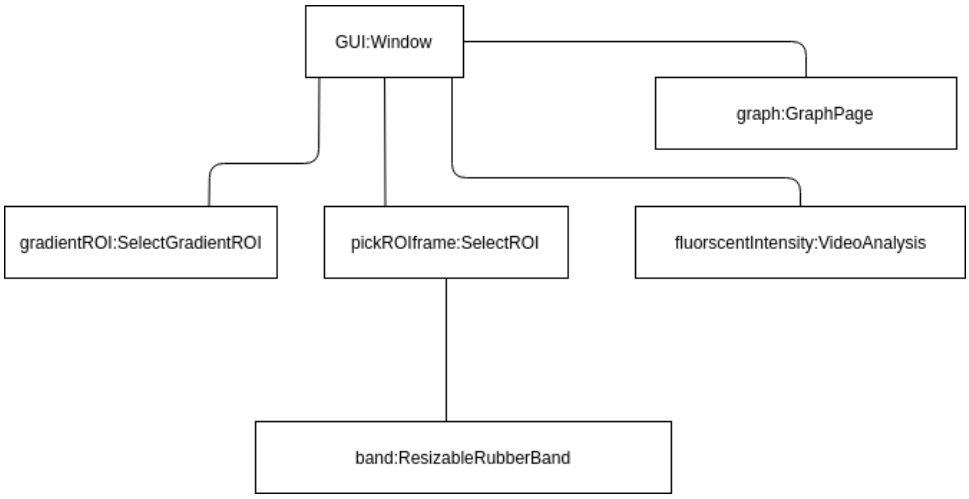


*Figure 58: This is the object diagram of the software, each box is with this format (**name of instance of the class : name of the class**)*

This program is consisted of 6 classes in total. Each window is represented by one class and there are total of 4 windows. The main window (Figure 53 and 54) is represent by the "Window" class. The page to show user desire frame and region of interest selection (Figure 55) was represented by the "SelectROI" class. The page for time interval selection for the gradient prediction (Figure 56) is represented by the "SelectGradientROI" class. The page to display all the graph (Figure 57) is represent by the "GraphPage".

The "Window" class is the class to load in the mainWIndow.ui. All the widgets in the main windows are belong to this class which means this class stores all the user's selections in the combo box (21, 22, 23, 24, 25) or spin box (3, 4, 5, 8, 20). The "Window" class also contains all the functions to instantiate all the other classes except the "ResizableRubberBand" class. Instantiating the class will make the corresponding windows to show up. Therefore, one of the most crucial tasks of the "Window" class is to connect all the buttons to the corresponding function to allow the right windows to show up or the right process to start which makes the GUI functional.

The "SelectROI" class is responsible for displaying the user's selected frame of the video and instantiate the "ResizableRubberBand" class. The "ResizableRubberBand" class is the draggable and resizable rectangular selector (28) inside the "SelectROI" window and on top of the displayed frame. The main purpose of "ResizableRubberBand" class is storing the pixel coordination of the four corners (x1, x2, y1, y2) of the rectangular selector. The "Window" class will then obtain these pixel coordinations from the instance of the "ResizableRubberBand" for cropping all the frames for K-means clustering and Sobel edge detection.

The "GraphPage" class is the class to load in the graphPage.ui. This class is responsible to use the matplotlib PyQt4 supportive backend (matplotlib.backends.backend_qt4agg) to display all the resulting graphs. This class main purpose is to display the list of all the resulting graphs and let the user to see the corresponding graph when they click the name of the graph in the list.

6.5 PyQT Signal and Slot:

The communication between classes(window) is very important. For example, the main window needs to update the display of the x and y pixel coordination (4) or time interval (5) after the user select the frame ROI or the time interval from a different window. The “Window” object needs to be able to communicate with the “ResizableRubberBand” object or the “SelectGradientROI” object to know when to update the display or the corresponding values. One of the key features of Qt is the use of signals and slots to communicate between objects [40] and this is another crucial reason to change from Tkinter to PyQt4. When something of potential interest happens, a signal will be emitted. A slot is a Python callable. If a signal is connected to a slot, the slot will be called when the signal is emitted. If a signal isn’t connected, then nothing will happen. The code (or component) that emits the signal does not know or care if the signal is being used. In this software, a signal is linked to the close button of the “SelectGradientROI” window, the signal will be emitted when the user clicks the close button to close the window. Another signal is linked to some functions in the “ResizableRubberBand” object so the signal will be emitted when the user drags or resizes the rectangular selector. The “Winodow” object connects all these signals to the corresponding function to update the display of the x and y coordination and the time interval.

6.6 Multi-Threading:

The “VideoAnalysis” class is not a window. This class is responsible to loop along all the frames in the video and apply both the K-mean clustering and Sobel edge detection to each frame to obtain the total edge sharpness, fluorescent intensity ratio, large vessel fluorescent intensity and the exchange vessel fluorescent intensity. Then the data will be classified into sharp and blur set. All the resulting data are stored inside this class. Performing K-mean clustering to all the frames in the video is a relatively time-consuming task. Therefore, threading is crucial for this task.

Qt and most GUI based applications are event based which means that execution is driven in response to user’s interaction [45]. In an event-driven application, an event will be created by clicking a button. This event will then be subsequently handled by the application to produce the expected output. Events are constantly pushed onto and remove from an event queue and processed sequentially. The event loop is started by calling .exec_() on the QtGui.QApplicationobject. This event loop runs with the same thread as the Python code. This thread which runs the event loop is commonly called the GUI thread which also handles all the window communication with the host’s operating system. Any event triggered by the event loop will also run synchronously within this thread by default. This means that window communication and GUI interaction will be frozen when the PyQt application is spending time on some other tasks in the code.

If all the tasks are very simple the GUI event loop take back control very quickly, this freeze will be imperceptible to the user. However, if longer-running tasks are needed to be performed, for example like applying K-means clustering to all the frames of the video, this long running task take up the tread and freeze the GUI event loop. The application will appear to be unresponsive. This solution is put this long running task into a new thread and keep the GUI thread free. PyQT provide a very straightforward interface to do multi-threading. This “VideoAnalysis” class inherits the QThread class. When the “Window” object instantiate this “VideoAnalysis” class and call the run() function, all those K-means clustering and Sobel edge detection would be performed in a separate thread. While looping along the video, a signal will be constantly emitted for the “Window” object to picked and call the function to update the progress bar. Therefore, two thread will be running simultaneously, one to loop along the video and obtain all the data and the GUI thread can update the progress bar at the same time.

6.7 PyInstaller

PyInstaller is a Python library that package a Python application and all its dependencies into a single package. The user can run the packaged application without installing a Python interpreter or any modules. PyInstaller works with Windows, Mac OS X, and Linux. However, it is not a cross-compiler: to make a Windows app you run PyInstaller in Windows; to make a Linux app you run it in Linux, etc. Gamez [2] is using a Microsoft Window computer, so a 32-bits python-3 virtual environment with PyInstaller and all the other was created to package the Python application so the software will work for both 32-bits and 64-bits Window.

When executing ***pyinstaller kevin_retinopathy_software.py***, the first thing PyInstaller does is to build a spec (specification) file, renamed as pyQTtrial.spec.


```
# -*- mode: python -*-

a = Analysis(['kevin_retinopathy_software.py'],
             pathex=['C:\\Users\\hk609\\Documents\\Thesis_Kevin_Retino_EXE_Final'],
             hiddenimports=[],
             hookspath=None,
             runtime_hooks=None)

ui_file = [('mainWindow.ui',
'C:\\Users\\hk609\\Documents\\Thesis_Kevin_Retino_EXE_Final\\mainWindow.ui', 'DATA')]

ui_file2 = [('graphPage.ui',
'C:\\Users\\hk609\\Documents\\Thesis_Kevin_Retino_EXE_Final\\graphPage.ui', 'DATA')]


pyz = PYZ(a.pure)
exe = EXE(pyz,
          a.scripts,
          exclude_binaries=True,
          name='kevin_retinopathy_software.exe',
          debug=False,
          strip=None,
          upx=True,
          console=False )
coll = COLLECT(exe,
               a.binaries,
               a.zipfiles,
               a.datas + ui_file + ui_file2,
               strip=None,
               upx=True,
               name='kevin_retinopathy_software')
```

Figure 59: pyQTtrial.spec content

A: The file location and name of the Python application

B: The file location and name of the main window ui file

C: The file location and name of the graph window ui file

D: Name of the final EXE file

E: Tell Pyinstaller to pack in the graph and main window ui files.

Then, execute the spec file by calling ***pyinstaller pyQTtrial.spec*** and a folder call dist with all the require library and a executable file in it.

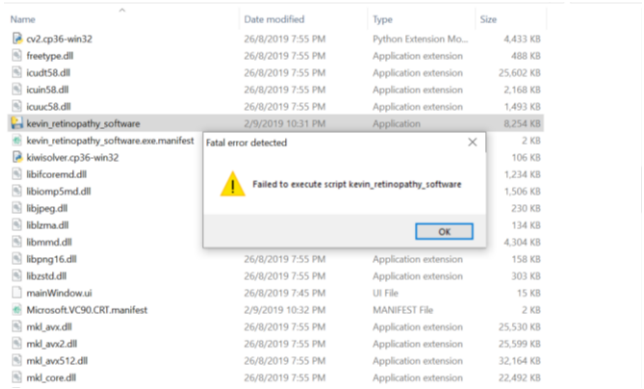


Figure 60:Demonstration of the error massage of the first time execution

An error was encountered on the first time to open the executable file. It was discover that the Pyinstaller didn't included all the required file for the skit-learn library to run. Then the missing files were then manually copied back into the skit-learn files in the Pyinstaller generated executable folder (Figure 61).

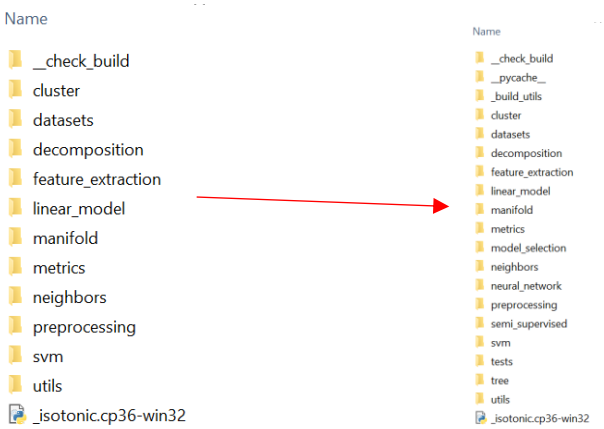


Figure 61: Show the changes did to the sklearn folder in the Pyinstaller packaged executable folder

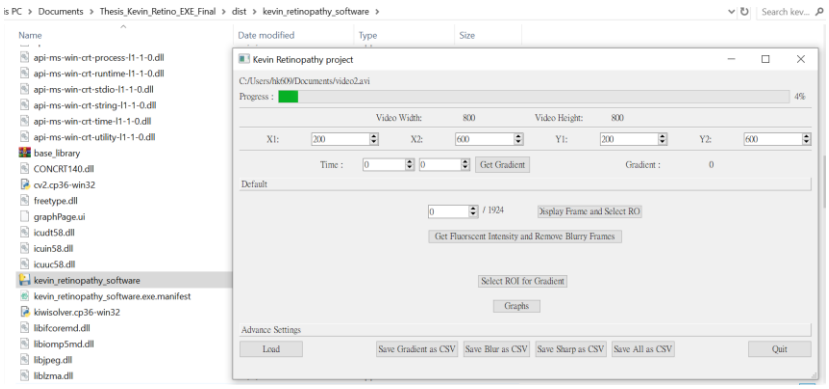


Figure 62: show the execution of the executable

Finally, the packaged application was executable. This software doesn't need installation any more, just copy the whole packaged folder to any Window computer and double click the application file, the main window should start up and run the whole program. This program has also successfully ran on Gamez[2] computer.

Chapter 7: Conclusion

7.1 Summary:

This project successfully wrote a piece of software base on openCV with python and some images and data processing techniques like K-means clustering and Sobel edge detection to allow automated but intelligent data capture from Sodium Fluorescein Angiography video to determine the solute flux. It can save researchers time and effort from the data capturing process. The data capture process now will be free from human error which will yield a more accurate solute flux.

This project firstly experimented on using K-means clustering to segment the exchange vessel groups and the large vessel group out of the FA frames to obtain the FIR for the FIR vs time graph. This project experimented on the two settings of K-mean clustering. One used random initial centers and the other one used the previous frame's centers found by K-means clustering as the initial centers of the K-means clustering for the current frame (Reuse center K-means clustering). The experiment found that the random initial centers K-means clustering output stable FIR when the maximum iteration was 7 or above and the best epsilon (specific accuracy) was 0.1. Maximum iteration below 7 cannot be used due to FIR vs time graph showed large amount of noise and severe deformation. Conversely, the reuse center K-means clustering showed no deformation and noise on the FIR vs time graph when the maximum iteration was 7 or below and a much shorter execution time than the random initial centers K-means clustering. Then, the difference on the gradient of the FIR vs time graph was further examine between the two K-means clustering setting. The random initial centers K-means clustering showed fluctuation on the gradient value when the maximum iteration was between 7 and 15. The reuse center K-means clustering showed either an ascending or descending trend on the gradient value when the maximum iteration was below 7 and the gradient value stabilized when the maximum iteration was between 7 and 15. Reuse center K-means clustering was decided to implement in the final software and maximum iteration 7 was set as default to prioritize gradient accuracy over the execution time, and allow user to lower the maximum iteration to reduce execution time.

This project then experimented on blurry frame classification by using Sobel edge detection. Convolution was performed with Sobel derivative operator on each FA frame to obtain an edge sharpness value. The edge sharpness versus frame number graphs were examined for all video and discovered a great separation between sharp and blurry frames in edge sharpness value. Sharp frames had higher edge sharpness and blurry frames had lower edge sharpness. A piece of code was created to loop along all the data points in edge sharpness vs frame number graph to classify sharp and blurry frames. The code firstly checked if the range of several neighboring data point (PtPbox) is larger than a specific value (tolerance value), then the data point needed a sharpness check, which take the mean of several neighboring data point (meanBox) and check is the current data point edge sharpness is lower or higher than the mean value. Lower means blurry frame and higher means sharp frames. A series of experiments were performed and the optimal value for PtPbox is 20, the meanBox is 10, the tolerance value is 0.1 and no histogram equalization is required. The sharp frames identification accuracy was above 80% and the blurry frames identification accuracy was above 96% for all the tested FA videos.

All these experimental codes were then connected by a graphical user interface based on python with PyQT4. Finally, the PyInstaller was used to package these Python codes into a stand-alone Microsoft Window executable for Gamez [2] to use.

7.2 Future Works:

- Implement a database into this software so the user doesn't have to save individual videos data as csv and open in Microsoft Excel. The user can just load in the video and extract all the data out of the video and directly store inside a database. The user can then see all the results inside one database rather than many separate data files. This data base can be a cloud database which the user doesn't have to worry about data lost and be able to access the data anywhere.
- Experiment on using other method than edge detection for blurry frame removal like convolution neuro-network and see whether can achieve an even higher accuracy on blurry frames removal.
- As python is a relatively slow language as it is a higher-level language, this software can be experimented on other lower-level language like C/C++ to further improve the run time of the program.
- Further examine the performance of other edge detection kernels like canny, Laplace, etc.

7.3 What have learnt from this project?

- Obtained some experience on object-oriented Python programming
- Obtained some experience with Python library like numpy, OpenCV and scikit-learn
- Obtained basic knowledge of digital images
- Obtained background knowledge of K-means clustering
- Obtained background knowledge of Sobel edge detection, image convolution and histogram equalization
- Obtained knowledge and experience on GUI programming, like multi-threading and PyQT signal and slot
- Obtained knowledge on packing Python code into Microsoft Window executable
- Obtained experience installing Linux package for python and creating and using Python virtual environment

Chapter 8: Bibliography

[1] Mrinalini Hoon, Haruhisa Okawa, Luca Della Santina, and Rachel O.l. Wong. 2014. Functional architecture of the retina: Development and disease. Progress in Retinal and Eye Research 42 (2014), 44–84. DOI:http://dx.doi.org/10.1016/j.preteyeres.2014.06.003

[2] Monica Gamez, Rebecca Foster, Simon Satchell, and Gavin Welsh. Therapeutically Targeting Heparan Sulfate To Restore Endothelial Glycocalyx During Diabetes. Ph.D Dissertation.

[3] Martin M. Nentwich. 2015. Diabetic retinopathy - ocular complications of diabetes mellitus. World Journal of Diabetes 6, 3 (2015), 489. DOI:http://dx.doi.org/10.4239/wjd.v6.i3.489

[4] T.A. Ciulla, A.G. Amador, and B. Zinman. 2003. Diabetic Retinopathy and Diabetic Macular Edema: Pathophysiology, screening, and novel therapies. Diabetes Care 26, 9 (2003), 2653–2664. DOI:http://dx.doi.org/10.2337/diacare.26.9.2653

[5] Michael J. Cree, John A. Olson, Kenneth C. Mchardy, Peter F. Sharp, and John V. Forrester. 1999. The preprocessing of retinal images for the detection of fluorescein leakage. Physics in Medicine and Biology44, 1 (1999), 293–308. DOI:http://dx.doi.org/10.1088/0031- 9155/44/1/021

[6] Emily Dawn Cole, Eduardo Amorim Novais, Ricardo Noguera Louzada, and Nadia K. Waheed. 2016. Contemporary retinal imaging techniques in diabetic retinopathy: a review. Clinical & Experimental Ophthalmology 44, 4 (2016), 289–299. DOI:http://dx.doi.org/10.1111/ceo.12711

[7] C. E. Baudoin, B. J. Lay, and J. C. Klein , 1984. Automatic detection of microaneurysms in diabetic fluorescein angiography. Rev Epidemiol Sante Publique, 32(3-4), pp.254-61.

[8] Roger Jagoe, John Arnold, Christopher Blauth, Peter L. C. Smith, Kenneth M. Taylor, and Richard Wootton, 1993. Retinal vessel circulation patterns visualized from a sequence of computer-aligned angiograms. Investigative Ophthalmology & Visual Science September, 34, pp.2881- 2887.

[9] M.Elena Martinez-Perez, Alun D. Hughes, Simon A. Thom, Anil A. Bharath, and Kim H. Parker. 2007. Segmentation of blood vessels from red-free and fluorescein retinal images. Medical Image Analysis 11, 1 (2007), 47–61. DOI:http://dx.doi.org/10.1016/j.media.2006.11.004

[10] Yalin Zheng, Jagdeep Singh Gandhi, Alexandros N. Stangos, Claudio Campa, Deborah M. Broadbent, and Simon P. Harding. 2010. Automated Segmentation of Foveal Avascular Zone in Fundus Fluorescein Angiography. Investigative Opthalmology & Visual Science 51, 7 (2010), 3653. DOI:http://dx.doi.org/10.1167/iovs.09-4935

[11] Yitian Zhao, Ian J.C. Maccormick, David G. Parry, Nicholas A.V. Beare, Simon P. Harding, and Yalin Zheng. 2015. Automated Detection of Vessel Abnormalities on Fluorescein Angiogram in Malarial Retinopathy. Scientific Reports 5, 1 (2015). DOI:http://dx.doi.org/10.1038/srep11154

[12] R.P. Phillips, P.G.B. Ross, P.F. Sharp, and J.V. Forrester. 1990. Use of temporal information to quantify vascular leakage in fluorescein angiography of the retina. Clinical Physics and Physiological Measurement 11, 4A (1990), 81–85. DOI:http://dx.doi.org/10.1088/0143- 0815/11/4a/309

[13] Russell P. Phillips, Philip G. Ross, Michael Tyska, Peter F. Sharp, and John V. Forrester. 1991. Detection and quantification of hyperfluorescent leakage by computer analysis of fundus fluorescein angiograms. Graefes Archive for Clinical and Experimental Ophthalmology 229, 4 (1991), 329–335. DOI:http://dx.doi.org/10.1007/bf00170690

[14] L. Martínez-Costa , P. Marco, G. Ayala, E.De Ves, J. Domingo, and A. Simó. 1998. Macular Edema ComputerAided Evaluation in Ocular Vein Occlusions. Computers and Biomedical Research 31, 5 (1998), 374–384. DOI:http://dx.doi.org/10.1006/cbmr.1998.1487

[15] H. Rabbani, M.J. Allingham, P.S. Mettu, S.W. Cousins, and S. Farsiu. 2015. Fully Automatic Segmentation of Fluorescein Leakage in Subjects With Diabetic Macular Edema. Investigative Ophthalmology & Visual Science 56, 3 (2015), 1482–1492. DOI:http://dx.doi.org/10.1167/iovs.14-15457

[16] Yitian Zhao et al. 2015. Automated Detection of Leakage in Fluorescein Angiography Images with Application to Malarial Retinopathy. Scientific Reports 5, 1 (2015). DOI:http://dx.doi.org/10.1038/srep10425

[17] Justis P. Ehlers, Kevin Wang, Amit Vasanji, Ming Hu, and Sunil K. Srivastava. 2017. Automated quantitative characterisation of retinal vascular leakage and microaneurysms in ultra-widefield fluorescein angiography. British Journal of Ophthalmology 101, 6 (2017), 696–699. DOI:http://dx.doi.org/10.1136/bjophthalmol-2016-310047

[18] Ayyakkannu Manivannan, Jarka Plskova, Alison Farrow, Sandra Mckay, Peter F. Sharp, and John V. Forrester. 2005. Ultra-Wide-Field Fluorescein Angiography of the Ocular Fundus. American Journal of Ophthalmology 140, 3 (2005), 525–527. DOI:http://dx.doi.org/10.1016/j.ajo.2005.02.055

[19] Yonatan Serlin et al. 2013. Novel Fluorescein Angiography-Based Computer-Aided Algorithm for Assessment of Retinal Vessel Permeability. PLoS ONE 8, 4 (2013). DOI:http://dx.doi.org/10.1371/journal.pone.0061599

[20] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. 2001. Constrained K-means Clustering with Background Knowledge. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01), Carla E. Brodley and Andrea Pohoreckyj Danyluk (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 577-584.

[21] Paul S. Bradley and Usama M. Fayyad. 1998. Refining Initial Points for K-Means Clustering. In Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98), Jude W. Shavlik (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 91-99.

[22] Juntao Wang and Xiaolong Su. 2011. An improved KMeans clustering algorithm. 2011 IEEE 3rd International Conference on Communication Software and Networks (2011). DOI:http://dx.doi.org/10.1109/iccsn.2011.6014384

[23] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. 1997. An efficient k-means clustering algorithm. Electrical Engineering and Computer Science 43 (1997).

[24] Douglas. Steinley. 2006. K-means clustering: A halfcentury synthesis. British Journal of Mathematical and Statistical Psychology 59, 1 (2006), 1–34. DOI:http://dx.doi.org/10.1348/000711005x48266

[25] Till Sieberth, Rene Wackrow, and Jim H. Chandler. 2016. Automatic detection of blurred images in UAV image sets. ISPRS Journal of Photogrammetry and Remote Sensing 122 (2016), 1–16. DOI:http://dx.doi.org/10.1016/j.isprsjprs.2016.09.010

[26] Jahne, B., 2005. Digital image processing, Berlin: Springer.

[27] E. Ong et al. 2003. A no-reference quality metric for measuring image blur. Seventh International Symposium on Signal Processing and Its Applications, 2003. Proceedings. (2003). DOI:http://dx.doi.org/10.1109/isspa.2003.1224741

[28] Niranjan D. Narvekar and Lina J. Karam. 2009. A noreference perceptual image sharpness metric based on a cumulative probability of blur detection. 2009 International Workshop on Quality of Multimedia Experience (2009). DOI:http://dx.doi.org/10.1109/qomex.2009.5246972

[29] Anon, 2019. Color Correction. Documents.sessions.edu.

[30] Nixon, M. and Aguado, A., 2006. Feature extraction and image processing, Oxford: Elservier.

[31] Anon, 2019. K-Means Clustering in OpenCV — OpenCV 3.0.0-dev documentation. Docs.opencv.org.

[32] Gupta, S. and Mazumdar, S., 2013. Sobel Edge Detection Algorithm. International Journal of Computer Science and Management Research, 2(2).

[33] Vincent, R. and Folorunso, O., 2009. A Descriptive Algorithm for Sobel Image Edge Detection. Proceedings of Informing Science & IT Education Conference (InSITE).

[34] Rosebrock, A., 2019. Convolutions with OpenCV and Python - PyImageSearch. PyImageSearch.

[35] Anon, 2019. Image Gradients — OpenCV 3.0.0-dev documentation. Docs.opencv.org.

[36] Anon, 2019. OpenCV: Histogram Equalization. Docs.opencv.org.

[37] Anon, 2019. About. Opencv.org.

[38] Anon, 2019. tkinter — Python interface to Tcl/Tk — Python 3.7.4 documentation. Docs.python.org.

[39] Anon, 2019. Riverbank | Software | PyQt | What is PyQt?. Riverbankcomputing.com.

[40] Anon, 2019. Support for Signals and Slots — PyQt v5.13 Reference Guide. Riverbankcomputing.com.

[41] Derpanis, K., 2010. Overview of the RANSAC Algorithm.

[42] Anon, 2019. OpenCV: Color conversions. Docs.opencv.org.

[43] Anon, 2019. Understanding K-Means Clustering — OpenCV-Python Tutorials 1 documentation. Opencv-python-tutroals.readthedocs.io.

[44] Podlozhnyuk, V., 2007. Image Convolution with CUDA.

[45] Summerfield, M., 2012. Rapid GUI programming with Python and Qt, Upper Saddle River, NJ: Prentice Hall.

Chapter 9 Appendices:

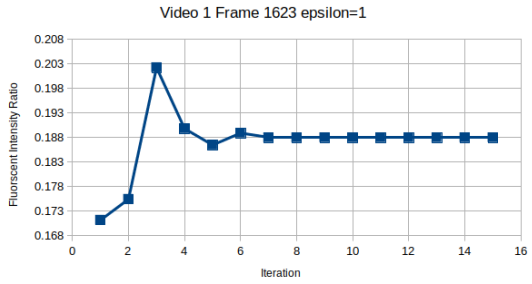
Video	Video Code from Gamez	Time frame used for slope (Seconds)
1	M1-1 10-26-18 11-15-34	817-626
2	m1-1 11-16-18 12-25-24	763-963
3	m1-1 11-2-18 16-20.53	600-800
4	m1-1 11-16-18 11-4-20	867-1067
5	B3 L base	419-618
6	m1-1 10-26-18 9-53-19	586-786
7	B2 L base	702-903
8	B3 LR W1	753-953
9	B2 LR W2	782-982
10	B2 O W1	918-718

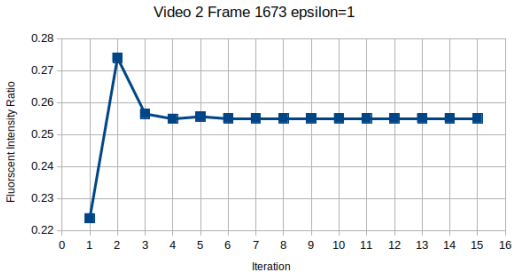
Gamez [2] provided data

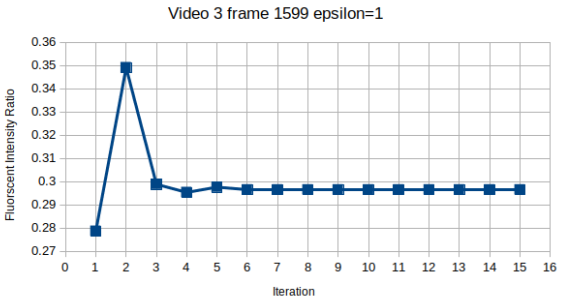
Video	Gradient obtained by Gamez manually	Gradient obtained by reuse centers K-means clustering with maximum iteration of 7
1	0.000338377	0.000308348505657042
2	0.00007524392035	0.000196957469354297
3	0.000225467253399	0.000279578588358473
4	0.000325593254117	0.000200973965889469
5	0.000481090911143	0.000450697830053718
6	0.000280486886397	0.000220745251645462
7	0.00011488	0.000160188654369653
8	0.000363566131583	0.00013585296488558
9	0.000228263271023	0.000229470562944612
10	0.000287632069897	0.000216260240757121

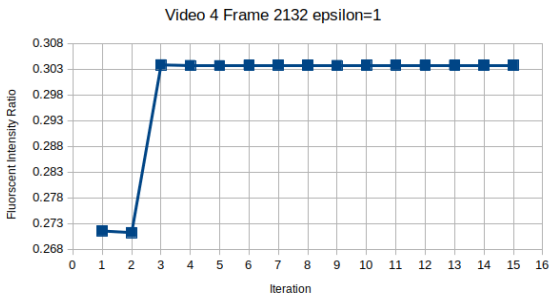
Figure 50 raw data

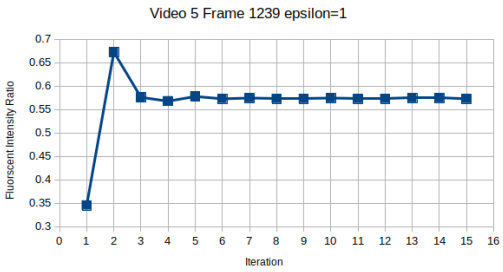
Random initial centers K-means clustering generated fluorescent intensity ratio vs iteration graphs when epsilon = 0.5 on 10 selected frames from all 10 video

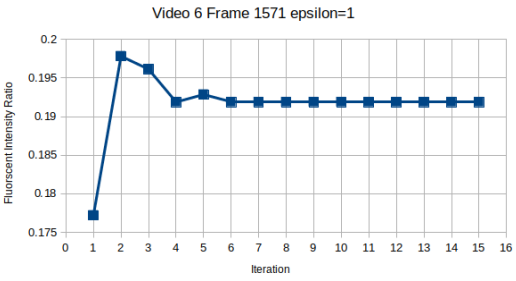


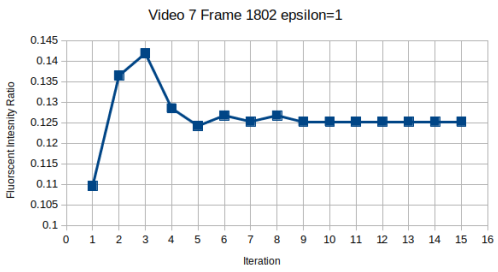


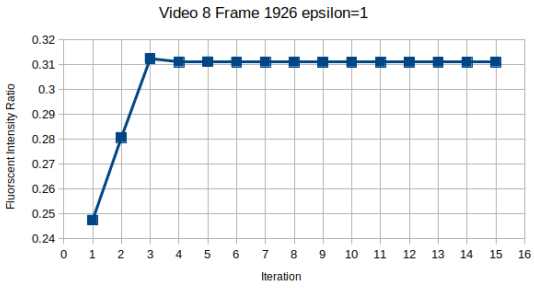


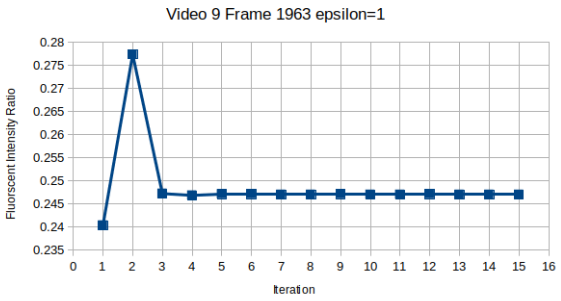


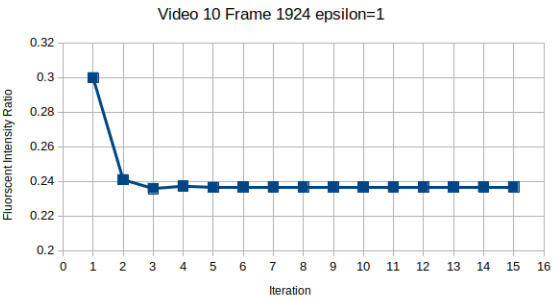




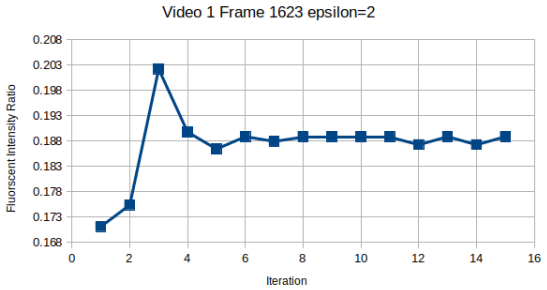


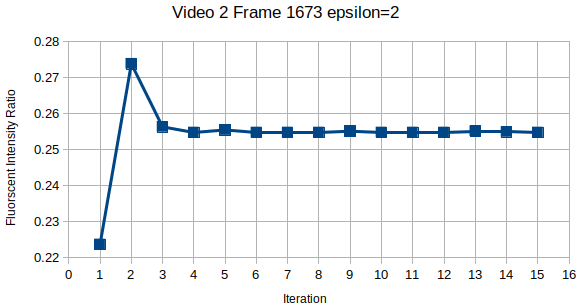


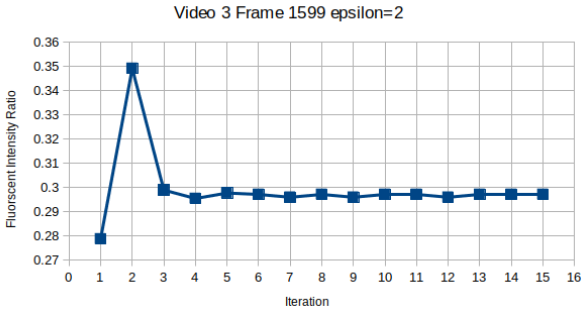


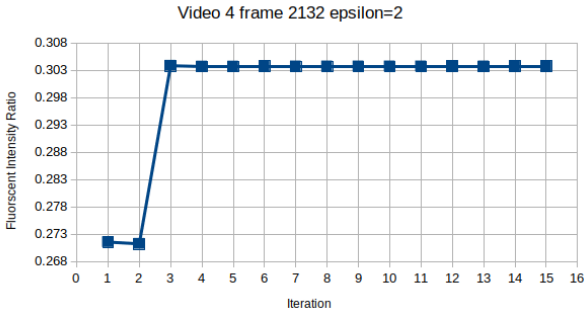


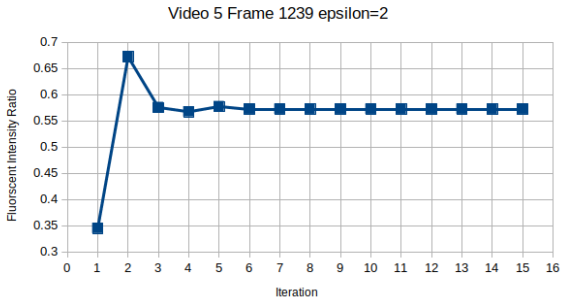
Random initial centers K-means clustering generated fluorescent intensity ratio vs iteration graphs when epsilon = 1 on 10 selected frames from all 10 video

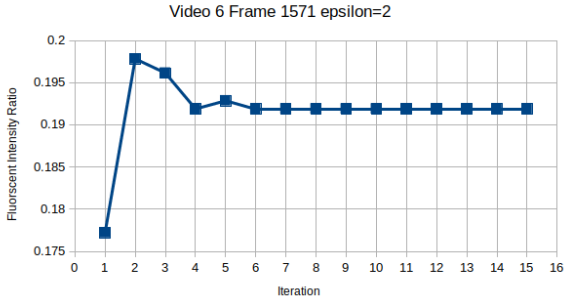


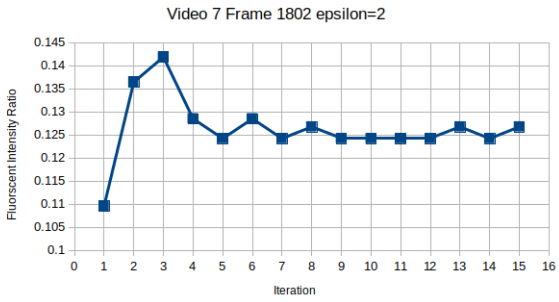


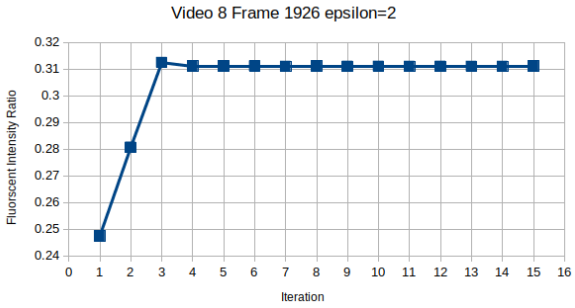


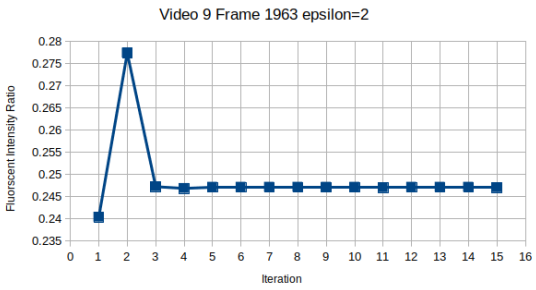


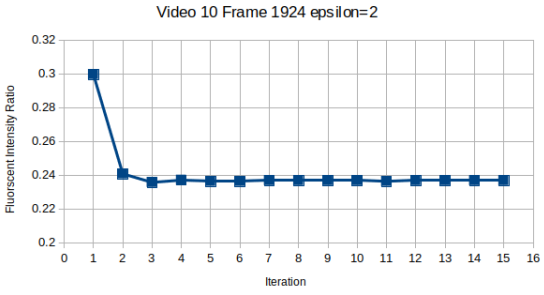












Random initial centers K-means clustering generated fluorescent intensity ratio vs iteration graphs when epsilon = 2 on 10 selected frames from all 10 video