KING'S College LONDON

University of London

AI-Assisted Visual Guidance for Endoscopic Robot

Final Project Report

Author: Yeonsu Kim

Supervisor: Dr Hongbin Liu

Student ID: 1346582

April 8, 2019

Abstract

This project aims to develop visual guidance system for robotic mesh-worm designed for colonoscopy based on computer vision techniques. The system developed in this project used dark region appeared at further most region of colon as an index for direction that robot needs to move. System was designed with three stages those perform different computer vision techniques. Each stage was designed to work sequentially since two higher stages required output result produced from first stage to perform their tasks.

To segment dark region, system smoothed image with Gaussian and median filter. Dark region was then segmented by thresholding and contour detection. To track dark region, contour center of dark region was calculated. Orientation of dark region was also computed in order to measure orientation of mesh-worm robot inside organ for stabilization of its movement to obtain clear image. At the end of project, visual guidance system successfully tracked dark region appears in endoscope image. Successful rate on tracking dark region was 97.5%. Hence, project had shown potential of visual guidance system.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purpose of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Yeonsu Kim

April 8, 2019

Acknowledgements

I would like thanks to Dr Hongbin Liu, my supervisor who supported me to accomplish this project. His advice was essential while working on project.

A special thanks to my wife and family for always supporting me.

Contents

Chapter 1

Introduction

The number of endoscopy procedure in the United Kingdom is expected to grow about 44% by 2020 compare to 2015 [1]. To accept increasing number of endoscopy procedure, it is inevitable that medical centers need to increase their capability of endoscopy. However, most medical centers in the United Kingdom are already suffering with difficulties on current number of endoscopy procedures. According to the survey conducted by Shenbagaraj [2], main difficulties on performing endoscopy responded by medical centers were meeting wait time, and shortage on staff and facilities.

Problem of endoscopy is not limited only to difficulties that medical centers are experiencing. Current method of endoscopy is performed by inserting device known as endoscope through anus or mouth. Generally, this method is known to be very safe however there are still some potential risks that patients can suffer. These potential risks are side effects caused by sedation such as difficulty on breathing and sick feeling, damage to internal organ, and discomfort during endoscopy procedure [3].

Developing a solution to current endoscopy problems for both patients and medical centers is not an easy task. However, in order to accept increasing number of endoscopy procedure and provide better quality service to patient, optimal solution must be considered.

1.1 Motivation

Realistic and optimal solution for current difficulties of endoscopy that medical centers are experiencing would be increase number of facilities through further investment and raise more gastroenterologist by encouraging medical student to be trained for this field. However, solving problems experienced by medical centers is not an only solution to prepare for future demand of endoscopy. Risks of endoscopy should be minimized as well to provide better quality service to patients.

There has been an active movement on research of new endoscopy methods as an alternative to current endoscopy. It is believed that new technology can minimize potential risks of current endoscopy. Wireless capsule endoscopy is one example of new endoscopy method. It is already implemented by medical centers as an alternative to current endoscopy [3]. Virtual endoscopy is state-of-art technology that reconstructs virtual map of internal organ using computed tomographic (CT) image [4]. Variety types of robotic endoscopes also has been introduced to the field as an alternative to current endoscope. Liu et al. [5] presented robotic mesh-worm designed for colonoscopy that is able to do locomotive motion like earthworm in the colon. Shikanai et al. [6] also introduced robotic endoscope that is able to move in organ with rotational locomotion using helical fins.

Development of new endoscopy methods could be another optimal solution to solve current difficulties of medical center on endoscopy procedure. In addition, it could minimize risks of current endoscopy as well. Hence, continuous research on new endoscopy technology is desirable and successful result will make great contribution to both patients and medical center. Therefore, this paper will introduce artificial intelligence assisted visual guidance system designed for robotic mesh-worm developed by Liu et al. [5]. It is believed that successful application of visual guidance system will improve quality of robotic mesh-worm and support its implementation in future.

1.2 Aim and Objectives

The project aims to develop artificial intelligence assisted visual guidance system for robotic mesh-worm developed by Lie et al. [5]. Application of navigation system will drive mesh-worm robot during colonoscopy procedure by minimizing damage to internal organ, lead robot to destination, and obtain clear image of internal organ. Moreover, it aims to make robot to fully functional by itself without external control by human.

To navigate robot just relying on visual information acquired from camera, there needs to be some objects appear in image very often hence it can be used as an index for navigation system to make decision. Fortunately, in most frame of endoscopy procedure video, dark region which is furthermost region from location of camera was observed. Since dark region represents empty space where there is no wall, it was decided that following dark region will provide optimal path for mesh-worm robot to its destination.

For successful development of the navigation system, information on dark region has to be analyzed and processed with computer vision techniques to use it as indicator for direction to move. Therefore, there are objectives those must be achieved during development process. Objectives of the system are:

- System must work in real-time.
- System must provide right direction where robot needs to proceed.
- System must handle errors to prevent sudden stop during procedure.
- System must stabilize robot's orientation in colon to obtain stable images.

Purpose of endoscope is examination of internal organ to discover any possible diseases. Objectives that has been introduced are all necessary elements to obtain clear and stable image of internal organ while driving robot to destination.

The worm-robot will continuously move toward its final destination. Hence, if system is not able to process images obtained from video in real time, it will not be able to adjust its direction at right time leading malfunction of robot. To prevent this situation, system must work in real time. In addition, it is preferred robot to continue its motion rather than pause at location in any circumstance,. Since, environment of internal organ is unexpectable, chance of potential error is always existing. Hence, it is important for system to handle this error therefore preventing robot from shut down.

Internal colon is tubular shape therefore it is difficult for robot to stabilize its motion. If motion of robot is not stable, unclear images can be obtained from camera. Hence, it is important to measure orientation of robot to stabilize its motion during procedure and prevent sudden change of robot's orientation.

1.3 Structure of Project

Chapter 1. Introduction

The chapter delivers introduction, motivation of project, aim and objectives.

Chapter 2. Background

The chapter introduces brief summary about endoscopy, computer vision techniques implemented in project, and their application in similar research topic that is available. Moreover, reviews on researches conducted in similar topic to this project is introduced as well.

Chapter 3. Specification and Requirements

The chapter introduces development environment of project and requirements for the system. Since visual guidance system developed in this project consists three stages of computer vision, detailed requirement is presented for each stage of system.

Chapter 4. Design and Implementation

The chapter illustrates specific design of each stage of visual guidance system. Moreover, implementation of each stage is presented.

Chapter 5. Testing and Results

The chapter introduces test process of visual guidance system and presents result obtained from the test. In addition, analysis on test result is conducted as well.

Chapter 6. Evaluation

The chapter evaluates performance of system. Moreover, evaluation on performance of each stage compare to their requirements.

Chapter 7. Conclusion and Future Works

The chapter provides summary of project and further researches recommended to increase quality of the system in future.

Chapter 2

Background

2.1 Endoscopy

Endoscopy is a medical procedure to examine internal organ. Current method of endoscopy is performed by inserting device known as endoscope through anus or mouth [3]. There are various types of endoscopy depends on what part of body is examined. Colonoscopy and gastroscopy are popular types of endoscopy. Colonoscopy examines all area of large bowl while gastroscopy examines esophagus, stomach, duodenum [7][8].

2.2 Image Filtering

Image filtering is an image processing technique used to modify an image. Normally, filtering is performed to sharpen or smooth an image [9]. Image filtering method can be classified into non-linear type and linear type. Linear type filtering produces output by computing weighted sum of neighborhood pixels while non-linear type filtering computes output by taking mean value of neighborhood [10]. In this project, Gaussian filtering was used as linear type filtering and median filtering was used as non-linear type filtering.

2.2.1 Gaussian Filtering

Gaussian filtering is a one method of linear filtering that uses low-pass kernel [10]. To apply filtering to an input image, Gaussian kernel is convolved with an input image and terminates spatial frequency that is lower than implemented Gaussian kernel's standard deviation value. Each pixel value of Gaussian kernel is determined by its standard deviation and kernel size. Pixel value of Gaussian kernel at each pixel is expressed by following formula.

$$G(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(\frac{-(x^2+y^2)}{2\sigma^2}\right) \quad (2.1)$$

In equation 2.1, x represents horizontal distance from an origin of an image while y represents vertical distance from an origin. σ represents standard deviation value of Gaussian kernel. One characteristic of Gaussian kernel is that size of kernel have to increase as σ increases. Hence, optimal size of kernel should be six times larger than σ. When image is smoothed by Gaussian filtering, smoothing effect becomes stronger as σ increases [11].

2.1.2 Median Filtering

Median filtering is non-linear type filtering that computes output taking median value of neighborhood pixels. Compare to Gaussian filtering, median filtering is stronger at reducing noises. Especially, it has advantage at reducing salt-and-pepper noise which is white noise appear in an image [12]. Similar to Gaussian filtering, kernel is used for median filtering and degree of effect on input image depends on kernel size.

Application of median filtering in medical field has been observed in some researches related to project topic. Xiangjun et al. [13] implemented median filter as preprocessing method to remove cavity appears in CT Computed Tomography) images. Obukhove et al. [14] introduced preprocessing method to reduce noise in endoscopy images by using adaptive median filter.

2.2 Image Segmentation

Image segmentation is process of separating interest object from rest of the image. There are various segmentation methods in computer vision technology therefore it is important to make right decision while choosing segmentation method for desirable output. In this project, thresholding and contour detection were decided to be used as image segmentation methods.

2.2.1 Thresholding

Thresholding is simple image segmentation method that binarizes gray-scale image. Thresholding computes output value into 1 and 0 according to given threshold value. Thresholding can be explained with following equation [15].

$$I'(x,y) = \begin{cases} 1, & I(x,y) > T \\ 0, & otherwise \end{cases} \quad (2.2)$$

Where $I'(x,y)$ is output pixel intensity, $I(x,y)$ is input pixel intensity, and T is threshold value. (x,y) represents horizontal and vertical distances from origin of input or output image.

While applying thresholding to an input image, deciding optimal threshold value is tough task. To avoid complex task of computing optimal threshold value, researches have done on methods to automatically compute optimal threshold for a given input. Otsu's automatic thresholding [16] is popular method to find proper threshold value based on gray-level histogram of an input image. Beside Otsu's method, various thresholding algorithms those compute optimal threshold value were presented in field of computer vision [17].

Since thresholding provides simplest way of segmenting target object from an image, its usage in medical field is very popular. Natarjan et al. [18] presented system that detect tumor from MRI image based on thresholding. In this research, tumors were detected by thresholding after preprocessed an image with filtering and histogram equalization. Ilhan et al. [19] introduced similar research to Natarjan et al. [18]. However, the research used morphological operation and pixel subtraction for preprocessing then segmented tumor from MRI image with thresholding. Xingjun et al. [20] presented path planning algorithm for virtual endoscopy that implements thresholding technique to segment target objects from CT image to construct 3D path. The research computed threshold value suggested by Kim et al [21] that is specialized for medical images.

2.2.2 Contour Detection

Contour in computer vision is a continuous line or curve that is formed at outer edge of an object which joins pixels those share same intensities [22]. There exist multiple methods to detect contours from an image. The best example of contour detection technique would be active contour model or snake developed by Kass et al. [23]. This popular method draws approximate contours around target object then this drawn contour is pulled towards edges of object forming actual contour of an object.

For real-time system, there are various contour detection methods to use. Upgrade version of snake proposed by Jin [24] and Lefèvre et al. [25] showed potential of active contour model on detecting contours while their application in real-time system. OpenCV (Open Library for Computer Vision) also provides contour detection function based on border following algorithm that produces topological structure of an input binary image developed by Suzuki et al. [26] which is applicable for real-time system.

2.3 Object Tracking

Dark region appears in endoscopy image is important object visual guidance program. After original input is preprocessed and segmented in previous stages, system can obtain information about this region such as its location. Depends on location of dark region in an image, direction required for robot's movement can be estimated. Images passed into system are taken from robot's point of view. Therefore, it is possible to assume that location of dark region in an image is actual location of dark region relative to mesh-worm robot's position.

Tracking location of dark region provides horizontal direction where robot needs to move. Since robot does not have vertical motion, system does not need to provide vertical direction. To track dark region, contour center of dark region has considered as good target to track. In computer vision, contour center can be calculated based on Green's theorem [27] which represents relationship between vector field, region exist in xy-plane, and contour of region [28].

To give correct direction to robot based on tracking information of dark region, it is important to set range of pixel locations to decide direction in real world. For example, in situation illustrated in figure 2.1, decision on correct direction is ambiguous task.

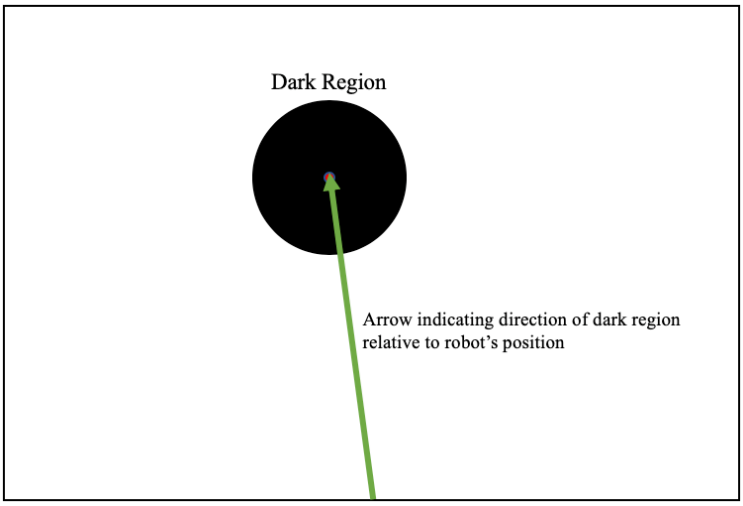

Figure 2.1: Illustration of case where decision on direction is ambiguous

Figure 2.1 shows two-dimensional image where dark region is appeared and arrow is drawn to estimate direction relative to robot's position at time when image is taken. In this case, since arrow is pointing in left it can be concluded that dark region exists on left side to robot hence system should order robot to move to left. However, it seems that order robot to move forward can also be right decision as well.

The easiest way to solve this problem is divide image frame into separate regions. Each divided region acts as an index for certain direction. For example, if size of image frame is 1000x720, regions can be divided into three regions which are two 400x720 size regions and a 200x720 region. Figure 2.2 shows illustration of the given example.

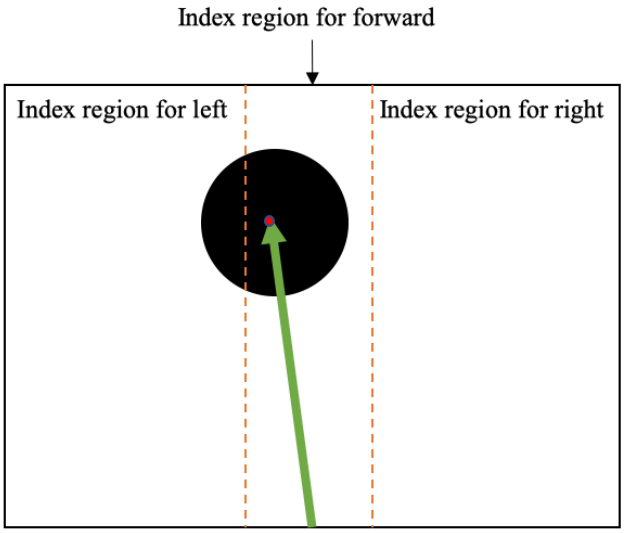

Figure 2.2 Illustration of method for direction index region division

By setting up separate regions as indexes for right, left, and forward direction, system can easily drive robot to desirable direction. Hence, system will drive robot to forward in case introduced in figure 2.2.

2.4 Object Orientation

To obtain stable endoscopy image, robot needs to do stable motion during its procedure. However, due to tubular structure of colon it is not an easy task for robot to do stable steering. Hence, there needs some control over robot's orientation to stabilize its motion during endoscopy. For this to be achieved, system should be able to compute accurate rotation of robot during its movement in colon.

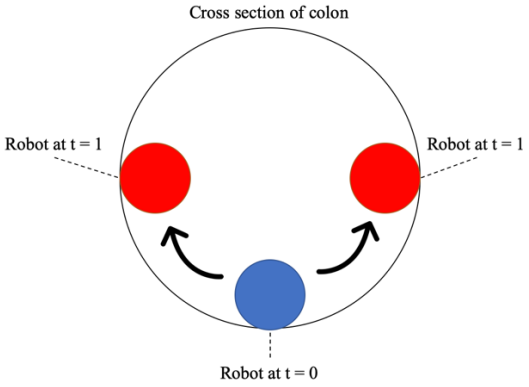

Figure 2.3: Undesirable motion of robot inside colon

Figure 2.3 shows the possible scenario where robot can lost its stable motion due to sudden change of its orientation. Due to tubular structure of colon, it is very difficult for robot to rotate more that 90 degrees in short time. However, if robot rotates near 90 degrees, it will cause robot to obtain unstable image due to sudden large amount of change in its orientation.

Optical flow is object tracking method of computer vision that compares previous frame and next frame of video. Specifically, optical flow discovers good target points to be tracked from previous frame and estimates motion of these target points in next frame [10]. However, optical flow is not suitable to be used as solution to calculate orientation change of robot. Instead, the idea of comparing previous frame and next frame is useful on obtaining rotation information of mesh-worm robot. By drawing linear shapes such as rectangle or straight line, orientation of robot can be computed by measuring angle difference between two consecutive frames.

2.5 Review on Similar Research Topic

Idea of using dark region as an index to navigate robotic endoscope was introduced by few researchers. Zhao et al. [29] proposed visual guidance system for wireless capsule endoscopy using dark region as navigation point. The research introduced segmentation algorithm and template matching algorithm where template matching algorithm is an alternative when segmentation algorithms fails. For segmentation, the research implemented watershed algorithm as a segmentation method. To minimize processing on unnecessary areas of input image, region of interest has set up as well.

Cho et al. [30] introduced autonomous driving algorithm for capsule endoscope using dark region. In this research, p-tile method has implemented instead of thresholding to binarize an image. It was explained from research p-tile method were used rather than thresholding since it is difficult to set specific threshold value for endoscope image. After binarizing image, dark region was obtained with morphology. Robot was driven by calculating distance between capsule endoscope and dark region.

According to similar researches introduced, using dark region as a navigation point to move robot was successful. Segmentation algorithm developed by Zhao et al. [29] showed average successful rate of 96.4% when algorithm was tested on intestine-like hose. Algorithm presented by Kim et al. [30] also showed higher successful rate during simulation. Two algorithms used different segmentation method but they succeeded to obtain dark region and drive capsule endoscopy autonomously. Hence, the key for development of autonomous visual guidance system for worm-mesh robot should be accurate segmentation of dark region.

Chapter 3

Specification and Requirement

3.1 Development Environment

The visual guidance system for mesh-worm robot was developed with OpenCV (Open Source Computer Vision Library) using python programming language. OpenCV is an open source library for computer vision that is suitable to use while developing real-time system [31]. It supports multiple programming languages such as C++, Java, and Python. The choice of Python as a programming language was made after discussion with supervisor.

System has built and implemented using OpenCV on Anaconda IDE. Both IDE and OpenCV were installed on Mackintosh OS X. Specific versions of programs used for system development is illustrated in table 3.1.

Program	Version
OpenCV	3.4.2
Python	3.6.6
Anaconda	4.5.11

Table 3.1: Version of program used for development of program

3.2 Requirements

The visual guidance system is consisted with three stages where they perform different computer vision techniques. These stages are preprocessing stage, segmentation stage, and tracking and orientation stage formed in hierarchical level. Therefore, first stage must be performed prior to second and third stage in order for overall system to work properly. If first stage fails to process an image and passes inappropriate output to second stage, system will fail or produces unexpected result. For this reason, requirements of each stage were defined for successful development of visual guidance system.

3.2.1 Preprocessing Stage Requirements

Tasks of preprocessing stage are read input, conversion to gray-scale, and application of filtering to an input. For an input image to be processed in later stage, first stage of the system should produce output without any errors. In addition, output from first stage must be in good quality for later application of computer vision techniques applied in later stages. Therefore, requirements of preprocessing stage are:

- Stage must read video input and let user to know when input is missing.
- Stage must convert input to gray-scale and prevent passing non-gray-scale input.
- Stage must use optimal parameters for filtering to produce best result for later stages.

Since visual guidance system works in real-time, missing an input is not a desirable scenario to the system. If system lost input for a long time, there is a risk of robot being lost during endoscopy procedure. Therefore, input should be read correctly and system should alert user immediately when system failed at obtaining input.

In image processing, RGB image is not a best input to work with. In addition, computer vision techniques applied in second and third stage do not accept RGB inputs at all. Therefore, it is important to convert input as gray-scale to avoid errors.

The system uses median filtering and Gaussian filtering as its filtering methods. These two filtering methods have parameters to be chosen. Thus, it is important to choose optimal parameter values to produce best output that can enhance quality of output from later stages of system.

3.2.2 Segmentation Stage Requirements

Segmentation stage is responsible to extract dark region from an input preprocessed in first stage of system. Since segmented dark region will act as a key in next stage, segmenting clear shape of dark region is important during this stage. Requirements of segmentation stage are:

- Stage must apply proper thresholding methods that segment dark region.
- Stage must decide proper thresholding parameter.
- Stage must be able to only detect contour of dark region.

It was introduced in chapter 2 that the system will use thresholding and contour detection as methods to segment dark region from an image. When preprocessed input is passed to segmentation stage, thresholding is the first method applied to input since contour detection cannot be performed on gray-scale input. Therefore, image is binarized through thresholding and output is passed for contour detection.

During implementation, basic thresholding and Otsu's method are going to be tested. Hence, it is required for this stage to compare the result and decide which method produces proper segmentation result of dark region. In case of basic thresholding, it is also required to decide appropriate threshold value to use.

Contour detection will be performed on binarized image from thresholding. While detecting contour it is important to detect only contour of dark region. There is always chance that binarized image contains multiple objectives other than dark region. If multiple contours are passed to next stage, it can cause confusion.

3.2.3 Object Tracking and Orientation Requirements

In object tracking and orientation stage, detected contour of dark region have to be tracked in order to provide location of itself to mesh-worm robot. In addition, detected contours need to be used as parameter to draw feature that can provide orientation information of robot. To track dark region and calculate orientation of the robot, this stage calculates center of contour and use it as target point where robot needs to move. Requirements of object tracking stage are:

- Stage must be able to calculate center of contour at each image frame.
- Stage must be able to provide direction to robot based on position of contour center.
- Stage must be able to draw feature based on detected contour and calculate change in orientation of that feature.
- Stage must handle any errors that can occur caused by lose of dark region.

Input to the system is video captured in real-time from camera attached to mesh-worm robot. Therefore, it is important to calculate center of contour at every image frame. If the system fails to compute center of contour, this can lead to providing wrong directional information to mesh-worm robot.

Contour centers inform location where the dark region appears in an image. To build these centers as indicator to navigate the mesh-worm robot, system must provide actual direction to move from robot's point of view. This means according to location of contour centers, systems must decide if mesh-worm robot needs to

move right, left, or forward. Since robot moves at surface of internal organ, vertical direction is not required to be computed.

During endoscopy procedure, there are some moments where dark region does not appear in an image. In this case, there is a chance of system failure since it lost the target. Therefore, it is required to handle this type of situation and provide a solution to avoid system failure during endoscopy procedure.

Chapter 4

Design

4.1 Overall System Design

The system is designed with three stages those perform different computer vision techniques. In computer vision, it is impossible to perform different levels of techniques at same time. Therefore, each stage was designed to be executed in sequential order. The system designed first stage as preprocessing stage which performs low-level techniques, second stage as image segmentation which performs mid-level techniques that requires preprocessed input. Finally, third stage was designed as tracking and orientation stage since output from second stage is required to perform high-level techniques. Architecture of visual guidance system is illustrated in figure 4.1.

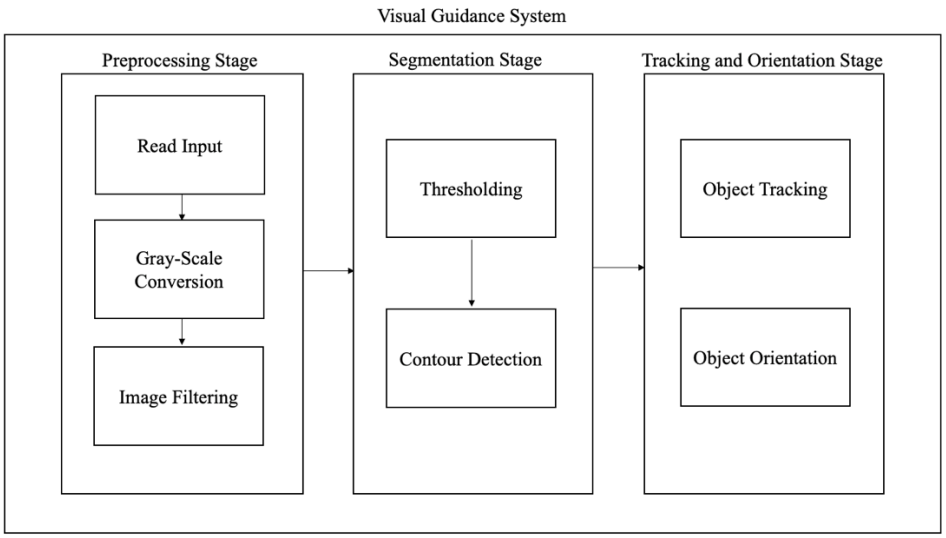

Figure 4.1: Architecture of visual guidance system

4.2 Preprocessing Stage Design

Program for preprocessing stage (see Appendix B.1) was designed and built separately to test and find optimal filtering parameter for given input. Since large program needs more time to analyze and modify, building separate program for preprocessing stage was essential to avoid complexity. In addition, preprocess program was only designed to compute image as an input. Figure 4.2 shows flow chart of preprocessing stage program.

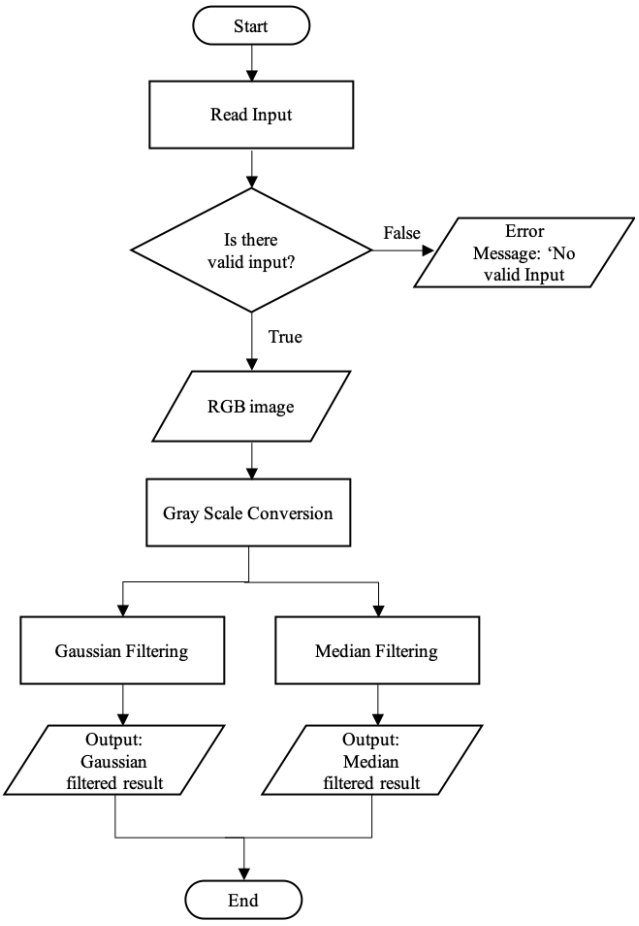

Figure 4.2: Flow chart of preprocessing stage

Two filtering methods were applied to the program using `cv2.GaussianBlur` and `cv2.medianBlur` function of OpenCV. To avoid missing input error, program was designed to send error message when there is no input. Reading input, converting RGB image to gray-scale, and filtering were designed to be performed in sequential order.

4.3 Image Segmentation Stage Design

Similar to preprocessing stage, separate program of segmentation stage (see Appendix B.2) was built to quickly analyze result and adjust parameters without complex tasks. Two tasks applied in this stage was designed to execute sequential order. Hence, thresholding performed prior to contour detection. Figure 4.3 shows flow chart of image segmentation stage program.

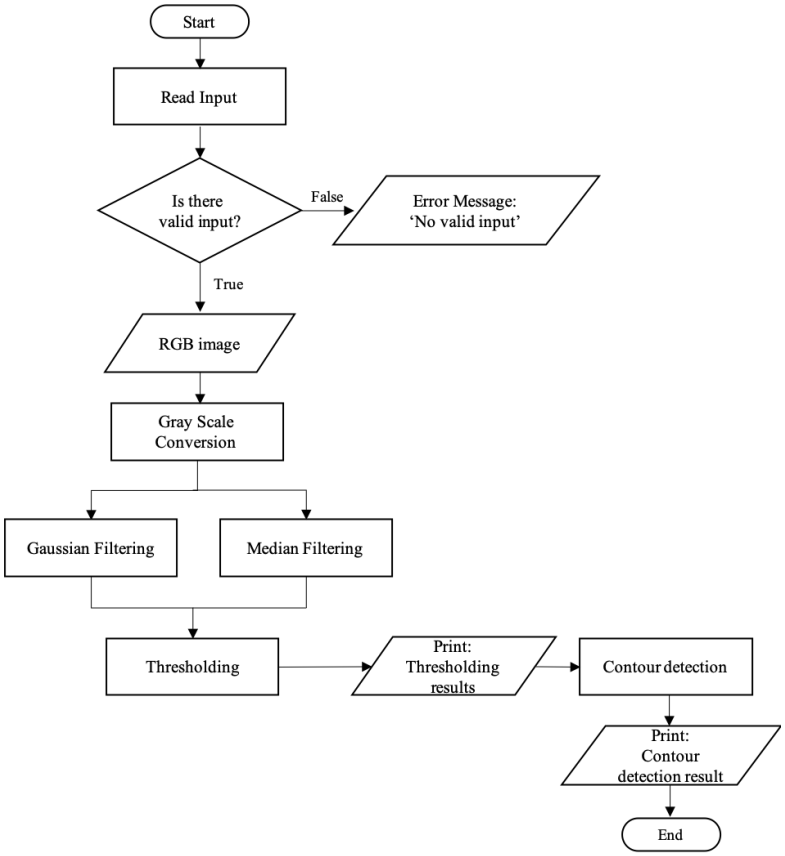

Figure 4.3: Flow chart of segmentation stage

Preprocess stage program was combined to segmentation stage program since all parameters for filtering was decided. As an output, program designed to produce thresholding result on Gaussian filtered image and median filtered image. Moreover, result of contour detection on thresholding result of Gaussian filtered and median filtered images were produced as well.

Techniques used in segmentation stage program were applied using functions from OpenCV libraray. Basic thresholding was applied by `cv2.threshold` function and Otsu's method was applied by adjusting type

parameter of thresholding function to `THRESH_BINARY+cv.THRESH_OTSU`. For basic thresholding, type parameter was set to `THRESH_BINARY_INV` which returns inverse output from equation 2.2. Contour detection was programed using function `cv2.findContours` then discovered contours were drawn using `cv2.drawContours` function.

4.4 Object Tracking and Orientation Stage Design

Object tracking and orientation stage requires output produced by prior stages. Therefore, new program designed by combining functions from first and second stage program for object tracking and orientation calculation which represents overall visual navigation system of project. Since two filtering methods are being tested, two versions of program were built which are version A that uses Gaussian filter and version B that uses median filtering. The figure 4.4 shows flow chart of object tracking and orientation stage program.

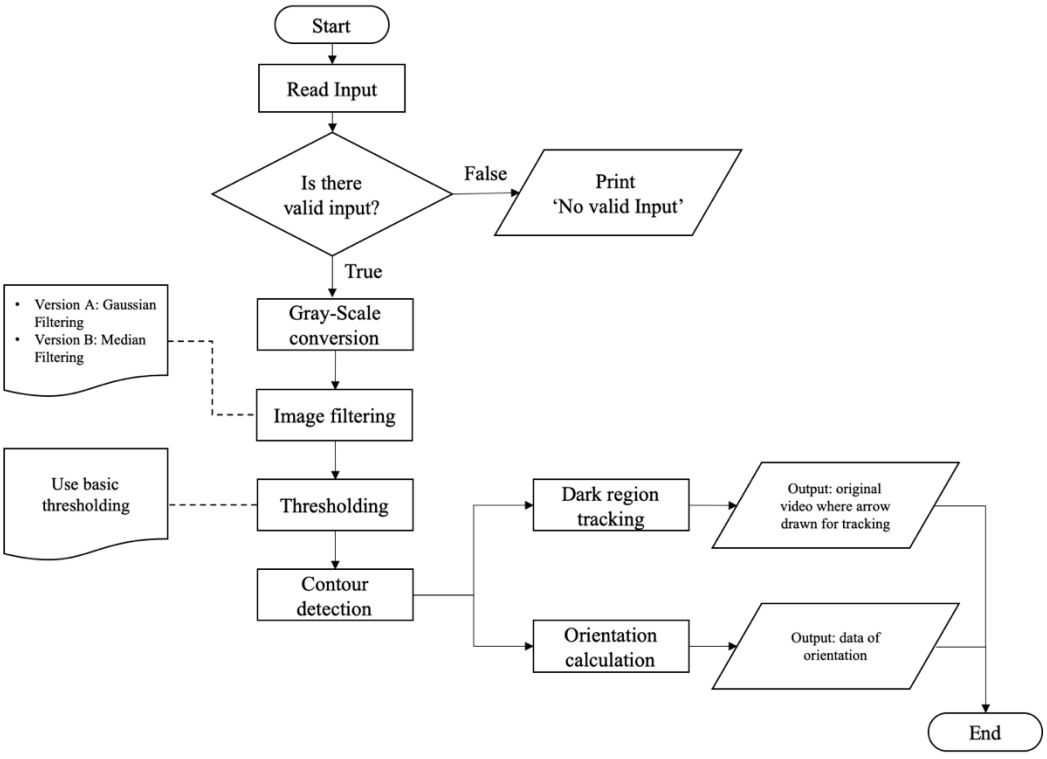

Figure 4.4: Flow chart of overall system

All functions are joined into single program and designed to produce two outputs. Object tracker produces video output which shows input footage from worm robot's camera where arrows are drawn to indicate dark region. Outputs from orientation calculation are data that includes change of orientation in between two frame and input footage where features for orientation calculation is presented.

Chapter 5

Implementation

5.1 Preprocessing Stage

5.1.1 Gaussian Filtering

Four Gaussian kernels of different sizes and standard deviations were chosen and applied to gray-scale images of both artificial colon and real colon. Size of kernel and standard deviation value of each Gaussian kernel are described in table 5.1.

Kernel Name	Standard Deviation (σ)	Size of Kernel
Gaussian_kernel_1	1	7 x 7
Gaussian_kernel_2	2	13 x 13
Gaussian_kernel_3	3	19 x 19
Gaussian_kernel_4	4	25 x 25

Table 5.1: Parameters of each Gaussian kernel

Since providing good quality output is important for next stage, filtering results on input images were analyzed to select best parameter. Figure 5.1 and 5.2 shows Gaussian filtering results of real colon and artificial colon.

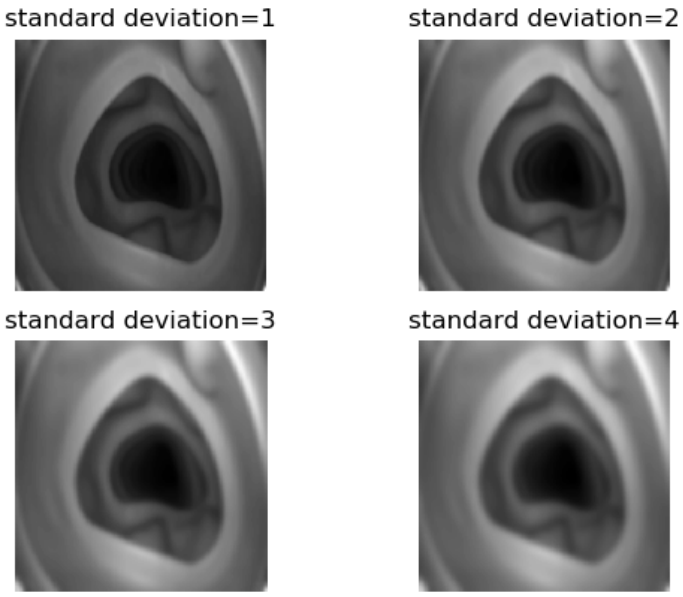

Figure 5.1: Gaussian Filtering results with different standard deviation value on artificial colon

Gaussian filtering on artificial colon showed stronger smoothing effect as standard deviation value increases. However, strong edges appear due to structural characteristics of artificial colon did not perfectly smoothed even with standard deviation of 4. Therefore, it was expected that increasing standard deviation would produce better smoothing effect on edges, but it also smoothed dark region which can cause difficulty in later stage.

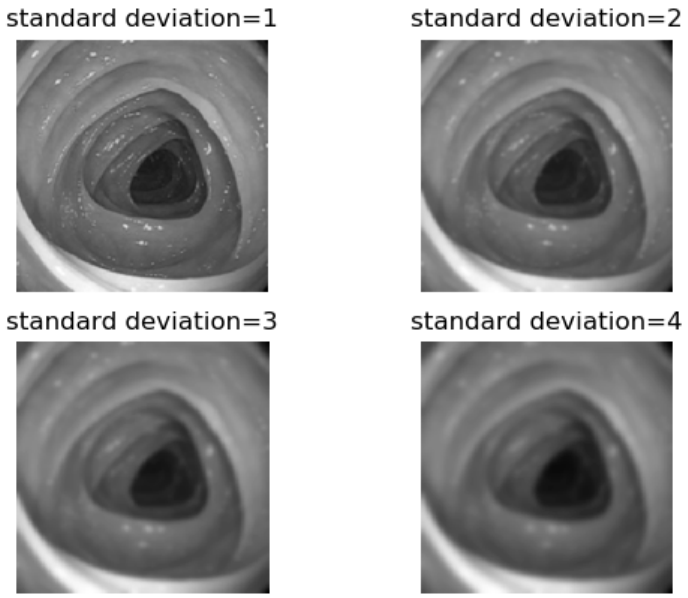

Figure 5.2**:** Gaussian Filtering results with different standard deviation value on artificial colon

Gaussian filtering showed better result when it was applied to image of real colon.

Since edges exists in real colon is not strong compare to artificial colon, filtering with standard deviation 3 and 4 produced satisfactory output that looks proper to be passed onto next stage. Comparing result of standard deviation 3 and 4, result obtained with standard deviation value of 4 was better at smoothing edges while highlighting dark region.

According to result obtained, it was decided that Gaussian kernel with standard deviation value of 4 would be the best output for later stages.

5.1.2 Median Filtering

Similar to Gaussian filtering, median filtering has implemented using different kernel sizes. After trials over different kernel sizes, best kernel size was decided. Since numerous numbers of different kernel sizes were tested, it was decided to show best result obtained during implementation. Figure 5.3 shows best result obtained of median filtering with kernel size = 19.

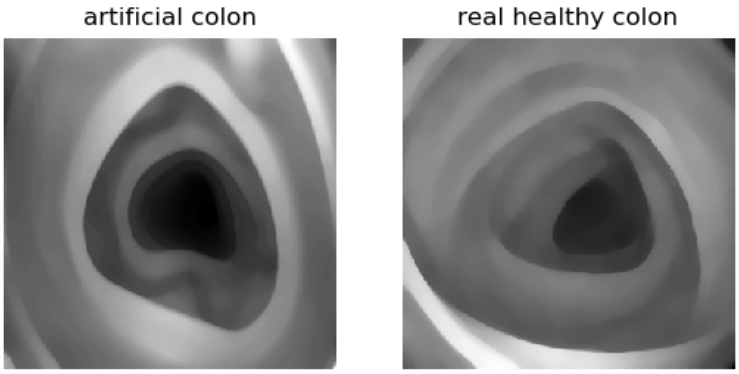

Figure 5.3: Median filtering image on artificial colon (left) and real colon (right) where kernel size = 19.

Median filtering on artificial colon did not showed good result compare to Gaussian filtering. Edges of artificial colon were emphasized rather than smoothed. However, dark area was emphasized very clearly compare to other areas of artificial colon. In case of real colon, result looked better than Gaussian filtering. Most of edges were smoothed and dark region was highlighted very well. Based on result in figure 5.3 and trials on different kernel sizes, it was decided to deliver result of median filtering conducted with kernel size of 19.

5.2 Image Segmentation Stage

5.2.1 Thresholding

Otsu's method did not required calculation of threshold value therefore there was no need to consider optimal threshold value. On the other hand, basic thresholding required to provide threshold value manually. There were two options to compute optimal threshold value which were using either method introduced by Kim [21] or manually testing different threshold values. Unfortunately, threshold value computed by Kim's method [21] did not produced satisfactory result. Therefore, numerous threshold values were tested manually to discover optimal threshold value for real colon and artificial colon images.

On real colon image, threshold value of 45 produced optimal result on segmenting dark region from. Figure 5.4 shows thresholding result with basic thresholding and Otsu's method.

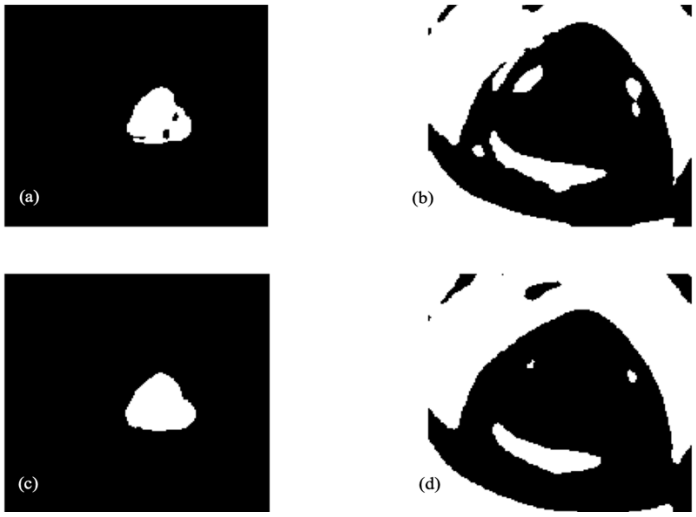

Figure 5.4: (a) Thresholding result on Gaussian filtered real colon image. (b) Otsu's method result on Gaussian filtered real colon image. (c) Thresholding result on median filtered real colon image. (d) Otsu's method result on median filtered real colon image.

Basic thresholding applied to Gaussian filtered input showed some holes inside of dark region but successfully segmented it from an image. Thresholding on median filtered image produced better result than thresholding result obtained with Gaussian filtered image. There was no noise found in thresholding result on median filtered image and clear shape of dark region was segmented.

Unfortunately, Otsu's method on both median and Gaussian filtered input was not successful at segmenting dark region. It failed to set threshold value that is optimal to only segment dark region hence showing

much broader dark areas appeared in input image. Based on result obtained by basic thresholding and Otsu's method, it was decided that use of basic thresholding is proper method to use for the visual guidance system during its procedure on real colon.

Thresholding result on artificial colon showed similar result to thresholding result of real colon image. For artificial colon, threshold value was set to 15. Figure 5.5 shows thresholding result on Gaussian and median filtered artificial colon image.

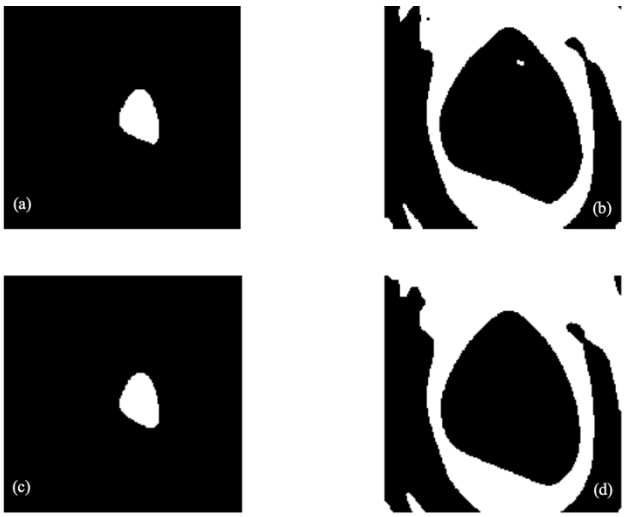

Figure 5.5: (a) Thresholding result on Gaussian filtered artificial colon image. (b) Otsu's method result on Gaussian filtered real artificial image. (c) Thresholding result on median filtered artificial colon image. (d) Otsu's method result on median filtered artificial colon image.

In figure 5.5, it was shown that basic thresholding was successful at segmenting dark region from input either it is filtered with Gaussian kernel or median kernel. Moreover, threshold result on Gaussian filtered image did not contained any holes unlikely to thresholding result of Gaussian filtered real colon image. On the other hand, Otsu's method showed disappointing result again. Threshold value was not set to optimal for extraction of dark region hence result obtained too much unnecessary objects. Based on result, it is again determined that basic thresholding is more suitable method for visual guidance system.

5.2.2 Contour Detection

According to decision on thresholding method, only results of basic thresholding result were passed to contour detection function. Using OpenCV function introduced in earlier of this section, contours in input images were detected and drawn on original RGB image of colon. Figure 5.6 shows detected contour drawn on real colon image.

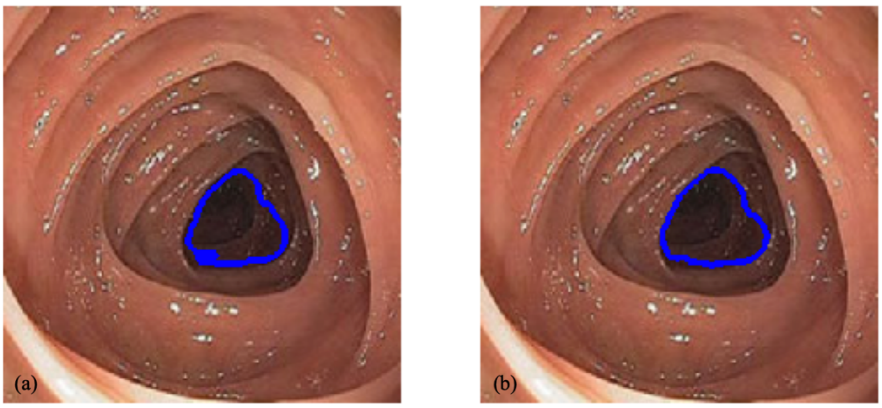

Figure 5.6: (a) Contour detection result of Gaussian filtered input. (b) Contour detection result of median filtered input.

On real colon, contour of dark region was detected much clearly from thresholding result of median filtered image. However, contour of dark region detection on thresholding result of Gaussian filtered image also showed almost identical result except there was bold shape observed at left-bottom corner of contour.

Contour detection result on artificial colon showed also pleasing result on detecting contour of dark region. Figure 5.7 shows contour detection result on artificial colon image.

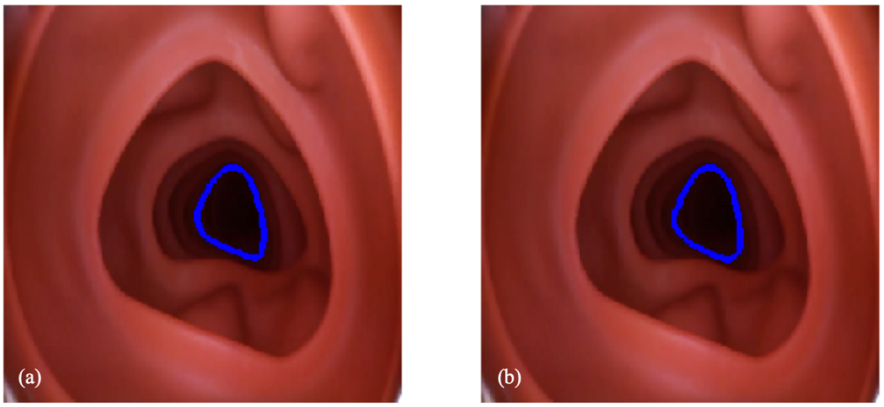

Figure 5.7: (a) Contour detection result of Gaussian filtered input. (b) Contour detection result of median filtered input.

Detection results on both thresholding result of Gaussian filtered image and median filtered image showed identical shape of detected contour. From figure 5.7, it is possible to observe that system has correctly detected dark region from an image.

5.3 Object Tracking and Orientation Stage

5.3.1 Object Tracking

Computation of contour center was done by calculating moments of dark region's contour. Moments of contour was computed using `cv2.moments` function from OpenCV library. Within moments obtained, contour center in x-axis and y axis were calculated within equation x and x [28].

$$center_{x_{axis}} = \frac{m_{10}}{m_{00}} \tag{5.1}$$

$$center_{y_{axis}} = \frac{m_{01}}{m_{00}} \tag{5.2}$$

Where m_{10}, m_{01}, and m_{00} are moments of contour. Obtained center of contour has been drawn to original input of the system.

Direction of dark region relative to mesh-worm robot's position inside internal organ was visualized by drawing arrows from x-axis center of image frame to center of contour. It was assumed that robot's current position at the time when image is taken is at center of frame therefore direction of arrow can be an indicator to navigate the robot.

Ranges of index regions for correct direction decision were computed by setting ratio of each regions to 4:2:4 where 4's are index region for right and left, and 2 is for forward movement. For example, if input image has vertical size of 1000, index region for left will be in horizontal range between 0 to 400 while index region for right will be in horizonal range between 600 to 1000.

5.3.2 Object Orientation

To measure orientation of robot, rotation angle of contour was calculated. Rather than angle relative to earth's surface, project focused on measuring angle between two sequential frame of input video to know orientation of robot. In order to have information on rotation of contour, rotatable rectangle was drawn to detected contour of dark region from previous stage. Rotatable rectangle was drawn to input image with `cv2.minAreaRect()` and `cv2.boxPoints()` function from OpenCV library [28].

For calculation of angle difference between two frames, information on rotation of rotatable rectangle was used. To judge whether there is large amount of angle orientation during robot's motion, threshold was set. Since sudden orientation near to 90 degree is not desirable for stable movement of robot, threshold was set to ±80 degrees. Hence system was set up to recognize robot's orientation is not stable if angle difference exceeds threshold by sending alert message.

Chapter 6

Testing and Results

6.1 Testing Environment

Both version A and B were simulated on video file of internal artificial organ taken by mesh-worm robot. The length of the video file was 128 seconds and its size was 1280x720. Test on video file of real endoscopy procedure was excluded because the movement of current endoscopy inside internal organ is very different from robotic mesh-worm. Therefore, it was concluded that testing on real endoscopy procedure would not show proper result where it can be used for further development.

6.2 First Test and Results

6.2.1 Testing: Version A

Version A is the visual guidance system that preprocess the input with Gaussian filtering. In first test, threshold value, kernel size, and standard deviation were identical to values introduced in chapter 4. Hence, parameters in of threshold value = 15, kernel size = 25x25, and standard deviation = 4 were used for testing. Figure 6.1 shows first object tracking result of version A

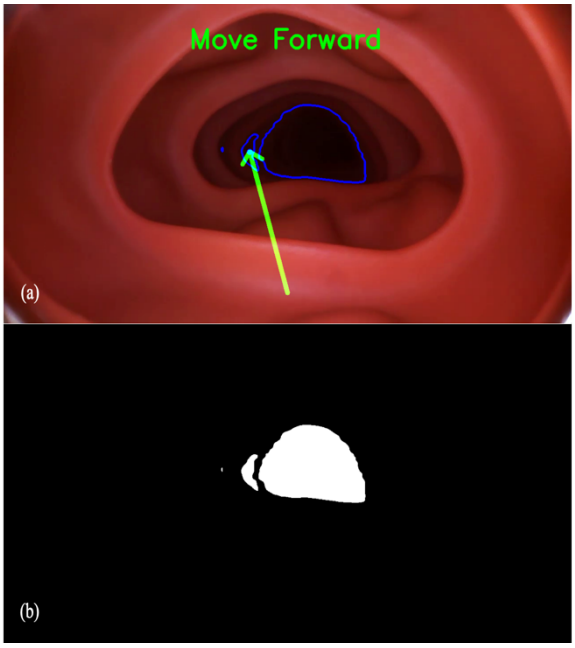

Figure 6.1: (a) Moment captured where system lost dark region during first test. (b) Thresholding result of same frame

On video file, parameters decided during implementation produced unexpected result. Rather than segmenting single object from an image, thresholding segmented multiple objects. Due to multiple objects segmented by thresholding, contours of all segmented objects were detected and caused confusion to system on finding contour center of dark region.

Throughout the first testing, similar result shown in figure 6.1 was observed and system was unsuccessful to tracking dark region in most of the video frames. There was also some moment that system detected dark region as can be seen in figure 6.2. However, overall performance of system on tracking dark region was disappointed.

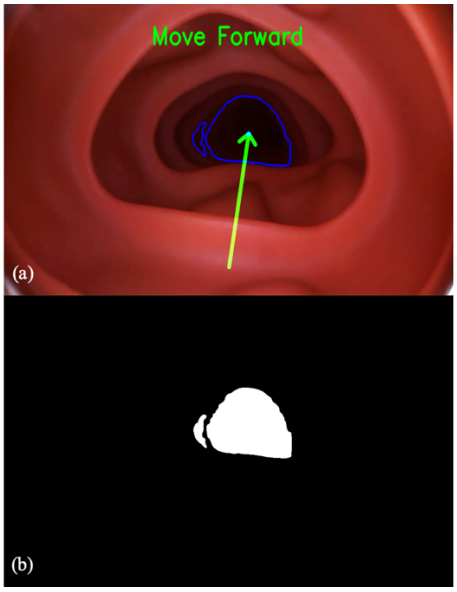

Figure 6.2: (a) First test result of version A where system succeeded to track dark region. (b) Thresholding result of same frame

Thresholding result on video file was also disappointed since parameters of first and second stage of system showed desirable results while they were tested on image file.

During first test on dark region tracking many problems were discovered. The main problem of the system was detection of multiple contours which required optimal solution to solve issues to produce expected result. Another problem was 'IndexError: list index out of range' occurred due to `cv2.moments()` function. As a reason for this error, it was assumed that detection of multiple contour is causing error. Hence, it was decided that fixing issues on multiple contour detection is priority than fixing an index error.

Test result on calculating orientation of contour was also disappointed as well. Since the system could not select optimal contour model, drawing rotatable rectangle to contour was unsuccessful.

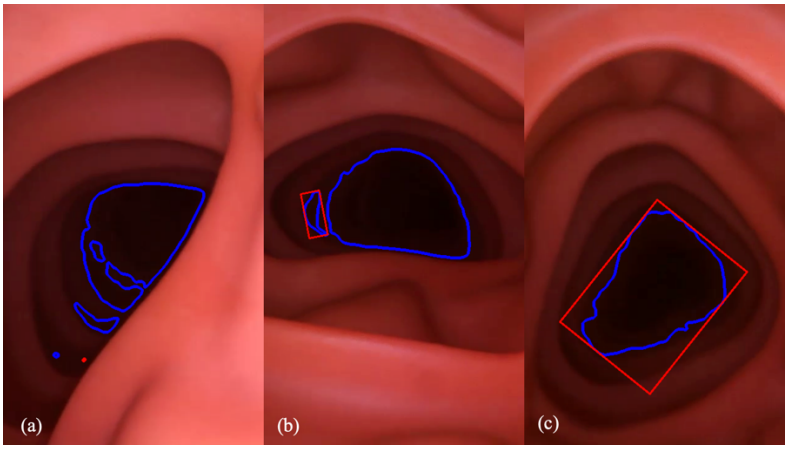

Figure 6.3: (a), (b) Unsuccessful test result of orientation calculation. (c) The moment when system was successful to detect dark region

Results shown in figure 6.3(a) and 6.3(b) was mostly observed during the test. System occasionally draw rotatable rectangle to contour of dark region as it can be seen from figure 6.3(c), but generally system failed and draw rectangle on other contours. Therefore, calculation of contour's orientation was meaningless since calculated result is mostly product of other contours.

6.2.2 Testing: Version B

First test of version B has performed in same environment. Median kernel size was set up to 19 which is the value decided in chapter 4. Since median filter showed good result during implementation it was expected at the beginning of test that median filter would produce better result. Unfortunately, test results of version B also showed similar result to version A. Figure 6.4 and 6.5 shows successful and unsuccessful result of locating contour center of dark region during first test.

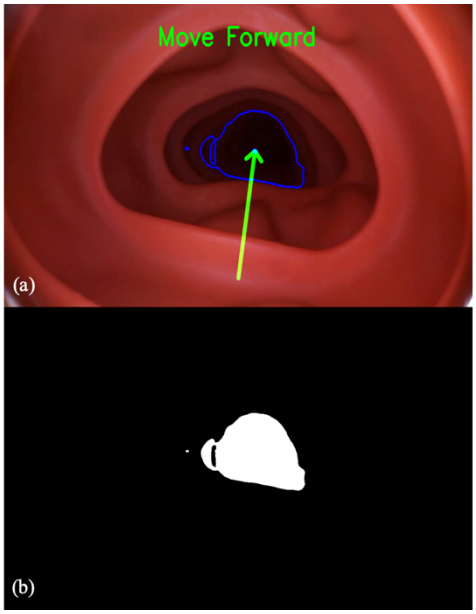

Figure 6.4: (a) Moment when system successfully detected dark region. (b) Thresholding result of same frame

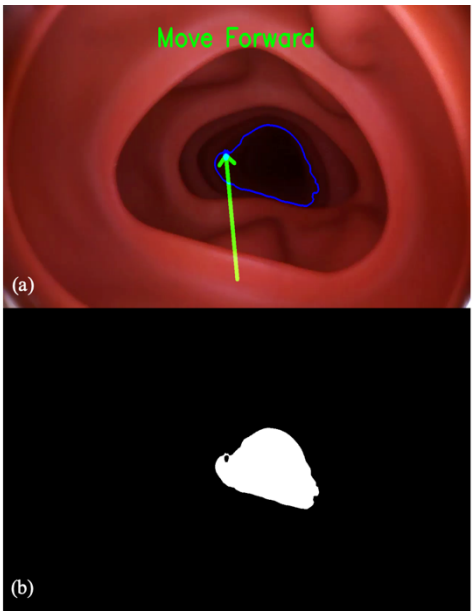

Figure 6.5: (a) Moment when system lost dark region. (b) Thresholding result of same frame

Thresholding segmented multiple objects or unclear shape of dark region most of the time during the task. Therefore, system mostly tracked another contour rather than contour of dark region. There was some moment that system successfully tracked contour of dark region, however it was not a desirable result that was expected before the test.

Since system failed to extract only the contour of dark region, calculation of contour orientation also did not work very well. While drawing rotatable rectangles to contour, similar result obtained in test of version A was observed. Figure 6.6 shows rotatable rectangles drawn during test of version B.

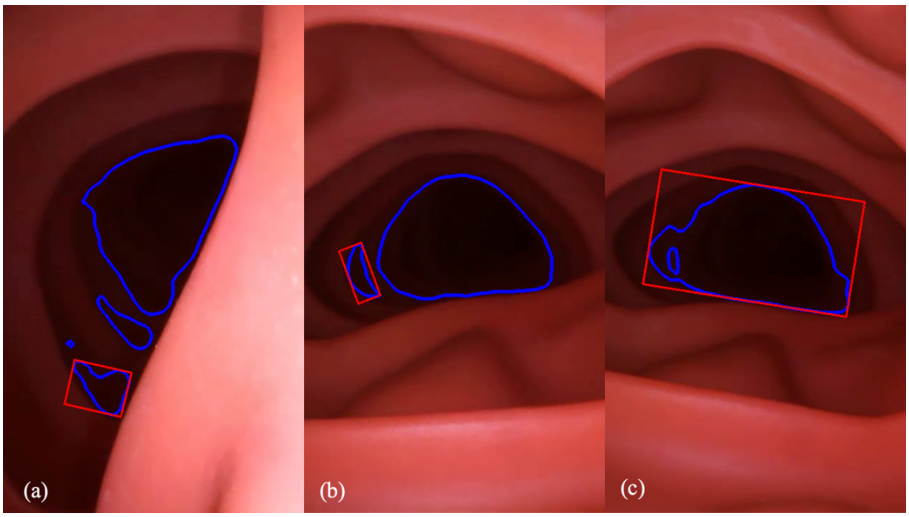

Figure 6.6: (a), (b) Unsuccessful test result of drawing rotatable rectangle for orientation calculation. (c) The moment when system was successful to draw rotatable rectangle on contour of dark region.

System was not able to draw rotatable rectangle on contour of dark region for most of the time during the test. Therefore, it was impossible to calculate orientation of dark region due to lack of information on its orientation.

6.3 Second Test and Results

In the first test, problems of version A and B were discovered. Both version A and B showed inability to choose right contour which represents dark area. In addition, multiple detection of contour was another problem that causes confusion on choosing correct contour to be tracked. These problems lead malfunction of object orientation task as well.

To solve issues of both versions discovered in first test, some parameter of `cv2.findContours()` function was adjusted. In first testing, retrieval mode of find contour function was set to `RETR_CCOMP` which returns every contour exists in an input image then classified them into two-level hierarchy [28]. Hence, it was discovered that reason for multiple detection of contour was due to wrong selection on retrieval mode and lack

of function to manage what contour is going to be used for the program. Therefore, retrieval mode was changed to `RETR_TREE` which detects every contour in an image but constructs full hierarchy.

However, even full hierarchy is obtained there is another problem that how contour of dark region can be recognized by the system. Since the system works in real-time, there was no time to pass parameter of dark region's contour to system for recognition. To solve this issue, contour areas of all detected contours were calculated. Fortunately, first test results showed that dark area had largest area compare to other segmented objects. Hence, array containing area of contours were created that stores data in descending order.

Two versions of system were named to version 2A and 2B after their modification in order to avoid any confusion. Their source code is available in appendix B.4

6.3.1 Testing: Version 2A

Parameters for preprocessing and segmentation stage did not changed from first test. Therefore, Gaussian kernel size = 15, standard deviation = 4, and threshold value = 15 were used for second test as well. Test was conducted on same video file that was used in first test. Figure 6.7 shows second test results.

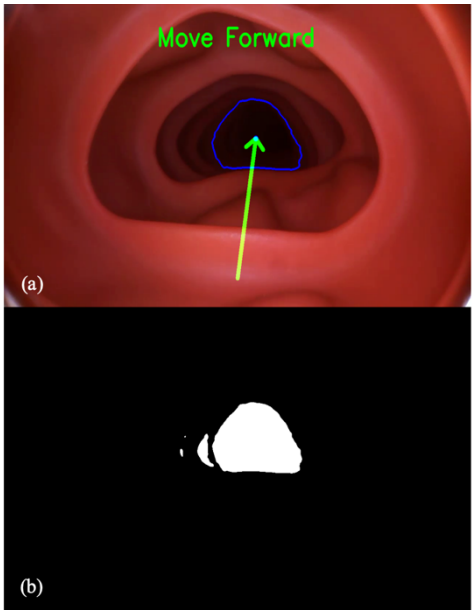

Figure 6.7: (a) Second test result of system. (b) Thresholding result of same frame

Results shown in figure 6.7 shows similar environment where multiple objects have been segmented by thresholding. In first test, version A failed at detecting only contour of dark region however version 2A was successful at detecting only dark region among multiple segmented objectives. In addition, center of contour was

located well since dark region was detected correctly. The successful rate of detecting dark region and computing contour center was very high except where the contour is disappeared. Figure 6.8 shows where missing contour is occurred.

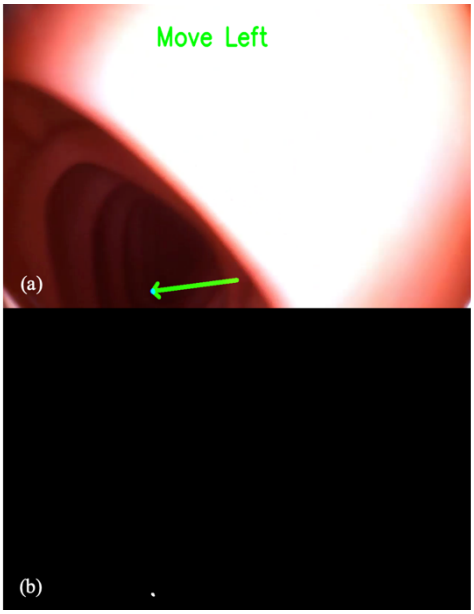

Figure 6.8 (a) Moment where contour is almost missing captured during test of version 2A. (b) Thresholding result of same frame

At point where contour is missing, thresholding result on filtered image showed that only tiny object is segmented. In addition, it also caused system to confront an index error by terminating the system. This phenomenon occurred at situation where too much light is reflected from surface of colon. However, it was interesting that center of contour was still able to be calculated even for tiny contour.

Since system was able to detect correct target contour, rotatable rectangles were drawn very successfully onto detected contour of dark region. As a result, orientation of this contour was also successfully calculated as well over the test until index error occurs.

5.3.2 Testing: Version 2B

Test of version 2B was proceeded in same environment as in first test. Figure 6.9 shows second test result of version 2B.

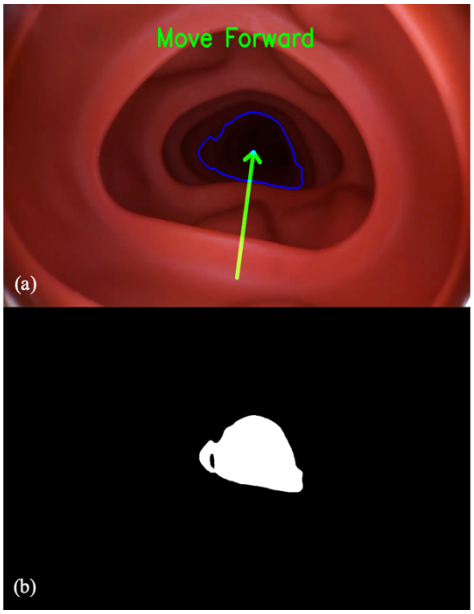

Figure 6.9 (a) Test result of version 2B. (b) Thresholding result of same frame

Captured frame of output was in similar environment to test result in figure 6.4 and 6.5. In first test, the system was unable to detect contour of dark region hence it computed center of inner contour appeared inside of contour of dark region sometimes. On the other hand, test result of version 2B showed that even there exists inner contour the system is able to detect contour that represents dark region.

On second test of version 2B, an index error occurred as well when system confront with very bright environment. Figure 6.10 shows moment where this error was occurred during test of version 2B.

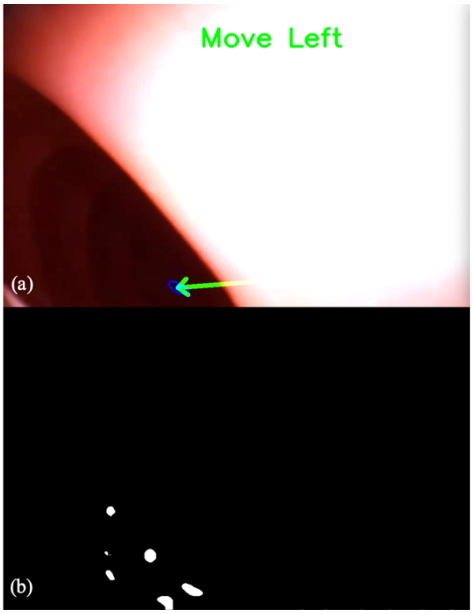

Figure 6.10: (a) The moment when system confronts with an error. (b) Thresholding value of same frame

Similar to situation where error occurred during test of version 2A, it has observed from captured image that thresholding produced very small objectives and output of system contained large reflection from surface due to light. Calculating orientation of contour was also successful during test of version 2A until index error occurs.

6.4 Final Test and Results

In second test, new problem has aroused and prevented the system to work successfully. This problem was 'out of index error' occurred by `cv2.drawContour()` function. To solve this issue in certain environment, exception to index error has been applied to both version 2A and 2B. Decision on applying exception to certain error was made because it was assumed that this situation can represents where robot cannot detect no dark region.

Within application of error exception to both version of visual guidance system, no more error occurred. To calculate successful rate of the system, it was assumed that robot was unable to detect dark region at certain frame if error occurred. Total frame number of video and number of frame where error occurred have counted to measure error rate on detection of dark region. Table 6.1 describes total number of frames of input video, total number of frame where error occurred, and successful rate on detecting and tracking dark region in final test.

	Version 2A	Version 2B
Total number of frames of input video	3754	3754
Total number of error occurred frame	93	93
Successful rate on dark region tracking	97.5%	97.5%

Table 6.1: Successful rate of system measured in final test

According to result introduced in table 6.1, both versions showed identical successful rate on dark region tracking. Compared to first result, system achieved successful progression for its application to robotic mesh-worm. Orientation of the contour was calculated well and produced desirable result that was expected at the beginning of test. Figure 6.11 shows brief calculation result of orientation of dark region.

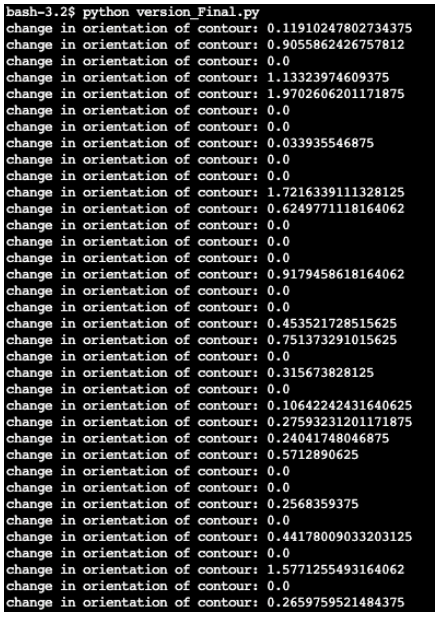

Figure 6.11: Calculation result of change in dark region's orientation

Result in figure 6.11 was obtained at the very beginning of input video file where robot is doing very stable motion. Therefore, there was no enormous change in orientation and it can be observed from result as well since computed change in orientation is near to 0.

Chapter 7

Evaluation

7.1 Requirements Evaluation

7.1.1 Preprocess Stage

In early stage of the project, it was required for preprocessing stage to read input, convert input to gray-scale, and provide optimal parameters to filtering methods for qualitive result of the system. Until final test of the system, there was no error on reading input or conversion to gray-scale. During implementation of the project, different values of filtering parameters were tested then optimal value was chosen based on test result. According to test result from image segmentation during implementation it was confirmed that decided filtering parameters were optimal choice for the system. This is also supported by the tracking result of the visual guidance system as well. Since final test result showed 97.5% on both version 2A and 2B, it is possible to conclude that filtering parameters decided in early stage of the project was optimal choice. Hence, preprocessing stage successfully managed requirement.

7.1.2 Image Segmentation Stage

The purpose of image segmentation stage was to extract dark region from an image through thresholding and contour detection. To achieve its purpose, it was required for image segmentation stage to decide best thresholding method, compute threshold value if decided method is basic thresholding, and detect only contour of dark region.

While stage was only working on image file, result obtained during implementation showed good result hence decision on what thresholding method to use was made very easily. In addition, optimal threshold value for both Gaussian filtered input and median filtered input was computed well through large number of trials on different threshold values. Results obtained with computed threshold value was confident as well.

Result of contour detection on image file was also good enough to satisfy with. Since most thresholding result was successful at segmenting only dark region, contour of dark region was able to be detected with simple process.

Unfortunately, both thresholding and contour detection of image segmentation stage struggled while working with video file as their input. During test of the navigation system, unexpected result was observed for both thresholding and contour detection. In most frames of video file, multiple objects were segmented after thresholding. These multiple segmented objects later confused contour detection. Due to multiple objects appear in binary image, the program suffered difficulty on only selecting contour of dark region since all contours were detected and drawn at first test of the system.

Overall, contour detection eventually succeeded to enable the system to choose only contour of dark region by calculating area of contour. However, it failed at removing 'out of index' error occurred due to `drawContours()` function. Rather than find solution to completely remove the error, exception to error was applied to the system which leaves potential of error occurrence during test in future.

Phenomenon that multiple objects are segmented by thresholding at some image frame of video did not solved well. Instead, these multiple objects were removed by drawing only contour of dark region using contour detection. Hence, it can be concluded that thresholding applied to the project produced mostly acceptable result but further research on better thresholding method is recommended.

7.1.3 Object Tracking and Orientation Stage

Through object tracking stage, the system was able to track dark region using contour center of dark region. Within the information about tracked contour center, approximate direction that robot needs to move was provided. At early stage of the project, requirements of object tracking stage were computation of contour center, provide direction to robot for its endoscope procedure, and handle error if robot lost dark region. The system achieved to calculate contour center and decide robot's direction to move. Handling error occurred when contour is lost was solved in previous stage of the system therefore it would be possible to assume that this requirement is fulfilled as well.

Even though object tracking stage was successful during test there were some difficulties that needs solution to improve quality of system. There were some moments where clear index about direction is required while giving direction to robot. Figure 7.1 shows captured image frame during test where decision on right direction is ambiguous.

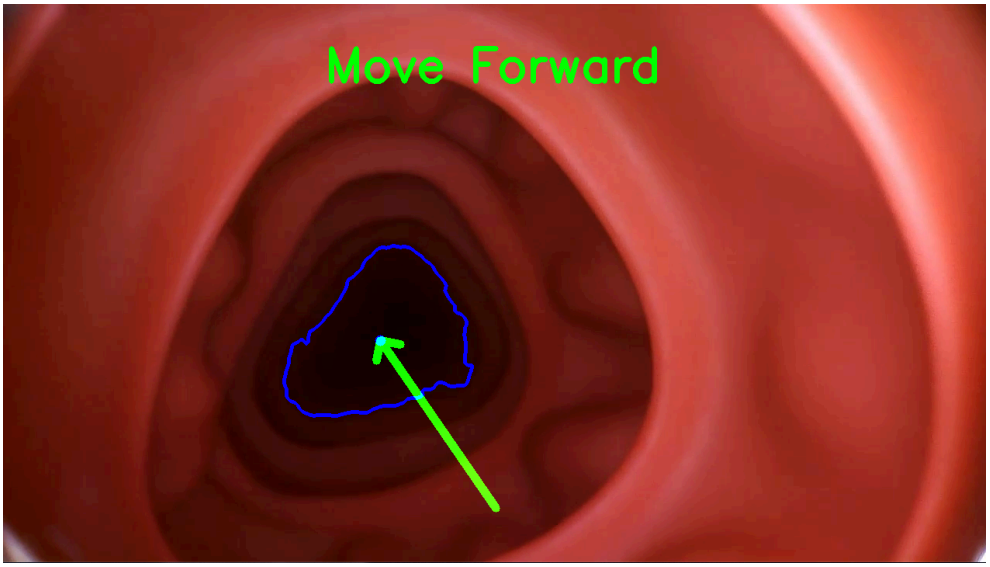

Figure 7.1: Ambiguous moment that is hard to decide proper direction to proceed

In case shown in figure 7.1, it was difficult to judge what direction is right for robot's movement. The dark region appears in an image seems to be located at left relative to position of robot at time when current frame is captured. However, it also seems okay for robot to move forward. When system confront with this type of ambiguous situation, there should be some index to make proper judgement.

7.2 Overall System Evaluation

Through several tests, system was able to discover serious issues those prevented its performance on endoscopy images. Especially problems discovered from first test result was shocking since preprocess and segmentation stage showed great performance when they worked with image files. However, working on video file was different from working on images hence it required much precise system.

Fortunately, system succeed to discover causes of problems and solve these issues with proper implementation of new function to the system. Overall, last test results showed high successful rate on tracking dark region appears in an image. In addition, orientation change of dark region between two sequential frame was also calculated successfully by providing estimation on robot's orientation change.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

Successful development of mesh-worm robot has potential to replace current endoscopy procedure. It is expected that implementation of mesh-worm robot in real medical field can solve current issues on endoscopy procedure of medical centers by minimizing time taken for procedure and required number of staffs. Moreover, it can be an optimal solution which can reduce risks accompanied by endoscopy procedure.

Through project, visual guidance system showed potential to navigate mesh-worm robot just rely on visual information obtained from camera. Within use of various computer vision techniques, the system was able to extract dark region appears in an endoscopy image and use it as indicator for navigation. In addition, change in orientation of dark region was also able to be calculated and provides way to estimate robot's orientation as well. Discovery of proper filtering methods and its parameter provided information on what choice should be made in tubular structure while preprocessing an image.

Undesirable results obtained from first test results showed that there needs to be much serious consideration on parameters and types of computer vision techniques while working with video or real-time input. There was no doubt that same techniques and parameters will produce different results from video and image files. Therefore, system has to suffer from malfunctions and unexpected errors.

Even there were difficulties during the project, these issues were handled very well and lead successful development of visual guidance system. In addition, valuable information on parameters of computer vision techniques that can be used as guidance in future study on endoscopy image was obtained during the project. To conclude, the project successfully developed visual guidance system and it is believed that successful application of system would be one optimal solution to solve current issues of endoscopy.

8.2 Future Works

The visual guidance system revealed some problems during its test on video file. Especially, segmentation of multiple objects as a result of thresholding was main problem that caused confusion to the system. The problem was handled by new contour detection methods however further study on solution to calculate optimal threshold value will minimize potential of detecting multiple threshold in endoscopy images.

While tracking dark region, correct judgement on direction of dark region relative to robot's position was required sometimes. System handled this issue by dividing image into three index regions for right, left and forward direction. However, it is still believed that method implemented to our system is not an optimal solution to clearly solve ambiguity in this type of situation.

Finally, even though change in orientation of dark region can provides estimated orientation of robot, it would be important to measure robot's orientation relative to earth's surface. Since obtaining clear image of internal organ is purpose of endoscopy, further research and development to stabilize robot's movement during its procedure would improve quality of the system and potential of success in future application.

References

[1] Brown H, Wyatt S, Croft S, Gale N, Turner A, Mulla A. Scoping the future: An evaluation of endoscopy capacity across the NHS in England. 2015. Available from: https://www.cancerresearchuk.org/sites/default/files/scoping_the_future_-_final.pdf [Accessed 4th March 2019]

[2] Shenbagaraj L, Thomas-Gibson S, Stebbing J, Broughton R, Dron M, JohnstonShaw D et al. Endoscopy in 2017: A national survey of practice in the U.K. *Frontline Gastroenterology*. 2019; 10:7-15. Available from: doi: 10.1136/flgastro-2018-100970

[3] National Health Service. *Endoscopy*. Available from: https://www.nhs.uk/conditions/endoscopy/ [Accessed 16th October 2018]

[4] Wood BJ, Razavi P. Virtual Endoscopy: Promising new technology. *Am Fam Physician.* 2002;66(1):107-113. Available from: https://www.aafp.org/afp/2002/0701/p107.html [Accessed 15th January 2019]

[5] Liu H, Arezzo A, Bernth J. A novel robotic meshworm with segment-bending anchoring for colonoscopy. *IEEE Robotics and Automation Letter*. 2017;2(3):1718-1724. Available from: doi:10.1109/LRA.2017.2678540

[6] Shikanai M, Murai N, Itoh H, Ishii A, Takanishi K, Tanoue K, et al. Development of a robotic endoscope that locomotes in the colon with flexible helical fins. *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 2009;5126-5129. Available from: doi: 10.1109/IEMBS.2009.5334579

[7] National Health Service. *Gastroscopy*. Available from: https://www.nhs.uk/conditions/gastroscopy/ [Accessed 16th October 2018]

[8] Cancer Research UK. *Colonoscopy.* Available from: https://www.cancerresearchuk.org/about-cancer/cancer-in-general/tests/colonoscopy [Accessed 17th October 2018]

[9] MathWorks. *What is Image Filtering in the Spatial Domain?*. Available from: https://uk.mathworks.com/help/images/what-is-image-filtering-in-the-spatial-domain.html [Accessed 10th November 2018]

[10] Szeliski R. *Computer Vision: Algorithms and Applications.* Springer; 2010. Available from: http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf

[11] Spratling M. *low level Artificial.* [Lecture] King's College London. 11th October 2018.

[12] OpenCV. *Image Smoothing*. Available from: https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html [Accessed 11th December 2018]

[13] Xiangjun G, Lianfang T, Lifei W, Zongyuan M. The Path Planning of Virtual Endoscopy Based on Image Segmentation. *2007 Chinese Control Conference*. 2006;567-570. Available from: doi: 10.1109/CHICC.2006.4347322

[14] Obukhove N, Motyko A, Pozdeevm A, Timofeev B. Review of Noise Reduction Methods and Estimation of their Effectiveness for Medical Endoscopic Images Processing. *2018 22nd Conference on Open Innovations Association (FRUCT)*. 2018;204-210. Available from: doi: 10.23919/FRUCT.2018.8468285

[15] Spratling M. *Segmentation Artificial.* [Lecture] King's College London. 8th November 2018.

[16] Otsu N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on System, Man, and Cybernetics*. 1979;9(1):62-66. Available from: doi:10.1109/TSMC.1979.4310076

[17] Kaur N, Kaur R. A review on various methods of image thresholding. *International Journal on Computer Science and Engineering*. 2011;3(10):3441-3443. Available from: http://www.enggjournals.com/ijcse/doc/IJCSE11-03-10-095.pdf [Accessed 15th January 2019]

[18] Natarajan P, Krishnan N, Kenkre NS, Nancy S, Singh BP. Tumor Detection Using Threshold Operation in MRI Brain Images. *IEEE International Conferences on Computational Intelligence and Computing Research.* 2012;1-4. Available from: doi:10.1109/ICCIC.2012.6510299

[19] Ilhan U, Ilhan A. Brain tumor segmentation based on a new threshold approach. *Procedia Computer Science*. 2017;120:580-587. Available from: doi:10.1016/j.procs.2017.11.282

[20] Xiangjun G, Lianfang T, Zongyuan M, Lifei W. The path planning of virtual endoscopy based on image segmentation. *Proceeding of 26th Chinese control conference*. 2007; 567-570. Available from: doi: 10.1109/CHICC.2006.4347322

[21] Kim DY, Chung SM, Park JW. Automatic navigation path generation based on two-phase adaptive region-growing algorithm for virtual angioscopy. *Medical Engineering & Physics*. 2006;28(4):339-347. Available from: doi: 10.1016/j.medengphy.2005.07.011

[22] OpenCV. *Contours: Getting Started*. Available from: https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

[23] Kass M, Witkin A, Terzopoulos D. Snakes: Active contour models. *International Journal of Computer Vision*. 1988;1(4):321-331. Available from: http://graphics.hallym.ac.kr/teach/2009/tcg/src/IJCV98Kass.pdf [Accessed 20th January 2019]

[24] Jin C. On Real Time Active Contours. *Proceeding of the 48th Annual Southeast Regional Conference Article*. 2010;47. Available from: doi: 10.1145/1900008.1900073

[25] Lefèvre S, Vincent N. Real Time Multiple Object Tracking Based on Active Contours. *Image Analysis and Recognition: ICIAR 2004.* 2004;606-613. Available from: doi: 10.1007/978-3-540-30126-4_74

[26] Suzuki S, Be KA. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing.* 1985;30(1):32-46. Available from: doi:10.1016/0734-89X(85)90016-7

[27] Khan Academy. *Green's Theorem*. Available from: https://www.khanacademy.org/math/multivariable-calculus/greens-theorem-and-stokes-theorem/greens-theorem-articles/a/greens-theorem

[28] OpenCV. *Structural Analysis and Shape Descriptors*. Available from: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#moments [Accessed 4th February 2019]

[29] Zhao Y, Lou Y. Vision Guided Navigation Based on Dark Regions and Template Matching for Capsule Endoscopy. *2013 IEEE International Conference on Information and Automation (ICIA)*. 2013;533-538. Available from: doi:10.1109/ICnfA.2013.6720356

[30] Cho H, Kim TJ, Lee JH, Kim HK, Park JO, Lee JH et al. Simulation Study of Autonomous Drive for Active Capsule Endoscopy. *2018 IEEE International Conference on Robotic Computing (IRC).* 2018; 403-406. Available from doi:10.1109/IRC.2018.00083

[31] OpenCV. *About*. https://opencv.org/about.html [Accessed 5th March 2019]

Appendix A

User Guide

A.1 System Requirements

The system has developed with python 3.6.6 version and OpenCV 3.4.2 version. For successful simulation, it is recommended for users to install equal or higher version of python and OpenCV that version used in development. Program is operational in Linux, OS X, and Windows operating system if OpenCV and python is installed.

A.2 Simulation Requirements

To avoid complexity while simulating the program it is required for user to place input files at same directory of program file. For version_A.py, version_2A.py, and version_Final.py, user is required to modify filtering methods by commenting out one of filtering methods coded in the program. For example, if user wants to simulate the program using Gaussian filter, medial filter should be commented out.

A.3 Simulation Method

To run program, open terminal and type

```
python (program file name.py)
```

If input file is video, type `-v (input_file_name)` after program file. For example, to run version_A.py with input video file test.avi, user should type

```
python version_A.py -v test.avi
```

to terminal. For an image file, read file using `cv.imread()` function coded in the program.

Appendix B

Source Code

B.1 Python Code for Preprocessing Stage

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

#Read Image
img_colon = cv.imread('Artificial_colon.png')
img_real = cv.imread('Real_Colon.jpg')

#Conversion to gray-scale
image_colon = cv.cvtColor(img_colon, cv.COLOR_BGR2GRAY)
img_real = cv.cvtColor(img_real, cv.COLOR_BGR2GRAY)
image_colon = cv.resize(image_colon, (300,300))
img_real = cv.resize(img_real,(300,300))

#Gaussian Filtering on artificial colon image
GBlur = cv.GaussianBlur(image_colon,(7,7),1)
GBlur2 = cv.GaussianBlur(image_colon,(13,13),2)
GBlur3 = cv.GaussianBlur(image_colon,(19,19),3)
GBlur4 = cv.GaussianBlur(image_colon,(25,25),4)

#Plot Result
plt.figure(1)
plt.suptitle('Guassian Filtered result: Artificial Colon Image')
plt.subplot(221),plt.imshow(GBlur,'gray'),plt.title('standard deviation=1'),plt.axis('off')
plt.subplot(222),plt.imshow(GBlur2,'gray'),plt.title('standard
deviation=2'),plt.axis('off')
plt.subplot(223),plt.imshow(GBlur3,'gray'),plt.title('standard
deviation=3'),plt.axis('off')
plt.subplot(224),plt.imshow(GBlur4,'gray'),plt.title('standard
deviation=4'),plt.axis('off')
plt.show()

#Gaussian Filtering on real colon image
GBlur_real = cv.GaussianBlur(img_real,(7,7),1)
GBlur2_real = cv.GaussianBlur(img_real,(13,13),2)
GBlur3_real = cv.GaussianBlur(img_real,(19,19),3)
GBlur4_real = cv.GaussianBlur(img_real,(25,25),4)

#Plot Result
plt.figure(2)
```

```
plt.suptitle('Guassian Filtered result: Real Colon Image')
plt.subplot(221),plt.imshow(GBlur_real,'gray'),plt.title('standard
deviation=1'),plt.axis('off')
plt.subplot(222),plt.imshow(GBlur2_real,'gray'),plt.title('standard
deviation=2'),plt.axis('off')
plt.subplot(223),plt.imshow(GBlur3_real,'gray'),plt.title('standard
deviation=3'),plt.axis('off')
plt.subplot(224),plt.imshow(GBlur4_real,'gray'),plt.title('standard
deviation=4'),plt.axis('off')
plt.show()

#Median Filtering on both artificial and real colon image
mBlur_colon = cv.medianBlur(image_colon,19)
mBlur_real = cv.medianBlur(img_real,19)

#plot result
plt.figure(3)
plt.suptitle('Median Filtered Result on real and artificial colon')
plt.subplot(121),plt.imshow(mBlur_colon,'gray'),plt.title('artificial
colon'),plt.axis('off')
plt.subplot(122),plt.imshow(mBlur_real,'gray'),plt.title('real colon'),plt.axis('off')
plt.show()
```

B.2 Python Code for Segmentation Stage

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

#Read Real Colon Image
Real_Colon = cv.imread('Real_Colon.jpg')
Real_Colon = cv.resize(Real_Colon,(300,300))
Real_Colon_sub = cv.imread('Real_Colon.jpg')
Real_Colon_sub = cv.resize(Real_Colon_sub,(300,300))

#Convert to gray-scale
Real_col_gray = cv.cvtColor(Real_Colon,cv.COLOR_BGR2GRAY)

#Read Artificial Colon Image
Art_Colon = cv.imread('Artificial_Colon.png')
Art_Colon = cv.resize(Art_Colon,(300,300))
Art_Colon_sub = cv.imread('Artificial_Colon.png')
Art_Colon_sub = cv.resize(Art_Colon_sub,(300,300))

#Convert to gray-scale
Art_col_gray = cv.cvtColor(Art_Colon,cv.COLOR_BGR2GRAY)

#Apply Filter
Real_Gaussian = cv.GaussianBlur(Real_col_gray,(25,25),4)
Real_Median = cv.medianBlur(Real_col_gray,19)
Art_Gaussian = cv.GaussianBlur(Art_col_gray,(25,25),4)
Art_Median = cv.medianBlur(Art_col_gray,19)

#Apply Thresholding to real colon image
ret1,real_thresh_gau = cv.threshold(Real_Gaussian,45,255,cv.THRESH_BINARY_INV)
ret2,real_thresh_med = cv.threshold(Real_Median,45,255,cv.THRESH_BINARY_INV)
ret3,real_otsu_gau = cv.threshold(Real_Gaussian,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
ret4,real_otsu_med = cv.threshold(Real_Median,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

#Plot thresholding result
plt.figure(1)
plt.suptitle('Thresholding Result of Basic thresholding and Otsu')
plt.subplot(221),plt.imshow(real_thresh_gau,'gray'),plt.title('Basic
Thresholding'),plt.axis('off')
```

```
plt.subplot(222),plt.imshow(real_otsu_gau,'gray'),plt.title('Otsu''s
mehtod'),plt.axis('off')
plt.subplot(223),plt.imshow(real_thresh_med,'gray'),plt.title('Basic
Thresholding'),plt.axis('off')
plt.subplot(224),plt.imshow(real_otsu_med,'gray'),plt.title('Otsu''s
mehtod'),plt.axis('off')
plt.show()

#Thresholding on artificial colon image
ret5,art_thresh_gau = cv.threshold(Art_Gaussian,15,255,cv.THRESH_BINARY_INV)
ret6,art_thresh_med = cv.threshold(Art_Median,15,255,cv.THRESH_BINARY_INV)
ret7,art_otsu_gau = cv.threshold(Art_Gaussian,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
ret8,art_otsu_med = cv.threshold(Art_Median,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

#Plot thresholding result
plt.figure(2)
plt.suptitle('Thresholding Result of Basic thresholding and Otsu on artificial colon
image')
plt.subplot(221),plt.imshow(art_thresh_gau,'gray'),plt.title('Basic
Thresholding'),plt.axis('off')
plt.subplot(222),plt.imshow(art_otsu_gau,'gray'),plt.title('Otsu''s
mehtod'),plt.axis('off')
plt.subplot(223),plt.imshow(art_thresh_med,'gray'),plt.title('Basic
Thresholding'),plt.axis('off')
plt.subplot(224),plt.imshow(art_otsu_med,'gray'),plt.title('Otsu''s
mehtod'),plt.axis('off')
plt.show()

#Contour detection on real colon
im1,real_cont_gau,hierarchy1 =
cv.findContours(real_thresh_gau,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)
im2,real_cont_med,hierarchy2 =
cv.findContours(real_thresh_med,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)
cv.drawContours(Real_Colon,real_cont_gau,0,(255,0,0),3)
cv.drawContours(Real_Colon_sub,real_cont_med,0,(255,0,0),3)

#Plot Result
plt.figure(2)
plt.suptitle('Contour detection result of real colon')
plt.subplot(121),plt.imshow(cv.cvtColor(Real_Colon,cv.COLOR_BGR2RGB)),plt.axis('off'),plt.t
itle('Gaussian Filtered')
```

```
plt.subplot(122),plt.imshow(cv.cvtColor(Real_Colon_sub,cv.COLOR_BGR2RGB)),plt.axis('off'),p
lt.title('Median Filtered')
plt.show()

#Contour detection on artificial colon
im3,art_contour_g,hierarchy3 =
cv.findContours(art_thresh_gau,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)
im4,art_contour_m,hierarchy4 =
cv.findContours(art_thresh_med,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)
cv.drawContours(Art_Colon,art_contour_g,0,(255,0,0),3)
cv.drawContours(Art_Colon_sub,art_contour_m,0,(255,0,0),3)

#Plot Result
plt.figure(3)
plt.suptitle('Contour detection result of artificial colon')
plt.subplot(121),plt.imshow(cv.cvtColor(Art_Colon,cv.COLOR_BGR2RGB)),plt.axis('off'),plt.ti
tle('Gaussian Filtered')
plt.subplot(122),plt.imshow(cv.cvtColor(Art_Colon_sub,cv.COLOR_BGR2RGB)),plt.axis('off'),pl
t.title('Median Filtered')
plt.show()
```

B.3 Python Code for Version A and B

```
import cv2 as cv
import numpy as np


# Read input
cap = cv.VideoCapture('Test.mp4')

#Array to store orientation of each frame
orient_ation = []
count = 0
orient_ation.append(count)

while True:
    ret,frame = cap.read()

    #Print error message if there is no input
    if ret == False:
        print('There is no valid input')
        break

    frame_gray = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)

    #frame_fil = cv.GaussianBlur(frame_gray,(25,25),4) #Comment out to simulate version_B
    frame_fil = cv.medianBlur(frame_gray,19) #Comment out to simulate version_A

    #Apply threshold
    ret4,frame_thresh = cv.threshold(frame_fil,15,255,cv.THRESH_BINARY_INV)

    #Detect Contours
    im2,frame_cont,hierarchy =.
    cv.findContours(frame_thresh,cv.RETR_CCOMP,cv.CHAIN_APPROX_NONE)

    #Draw contour on input image
    cv.drawContours(frame,frame_cont,-1,(255,0,0),3)

    #Compute moments of contour
    M_frame = cv.moments(frame_cont[0])

    #ignore frame when 'm00'=0
```

```
if M_frame['m00']==0:
        M_frame['m00']=1
cx_frame = int(M_frame['m10']/M_frame['m00']) #contour center in x-axis
cy_frame = int(M_frame['m01']/M_frame['m00']) #contour center in y-axis

#Draw contour center
cv.circle(frame,(cx_frame,cy_frame),7,(255,0,0),-1)

Draw arrow from center of frame to contour center
cv.arrowedLine(frame,(640,650),(cx_frame,cy_frame),(0,255,0),10)

#Index region for direction
left_index = int((4*frame.shape[1])/10)
right_index = int((6*frame.shape[1])/10)


#Direction index
if cx_frame <= left_index:

    cv.putText(frame,'Move Left',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

elif cx_frame >= right_index:

    cv.putText(frame,'Move Right',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

else:

    cv.putText(frame,'Move Forward',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

#Computes rotatable rectangle that fits to contour of dark region
min_rec = cv.minAreaRect(frame_cont[0])
orient_ation.append(min_rec[2])
rotation = abs(orient_ation[count]-orient_ation[count+1])
count = count+1

#Computes corner points for rotatable rectangle to draw it on original image
box = cv.boxPoints(min_rec)
box = np.int0(box)

#Draw rotatable rectangle to original image
cv.drawContours(frame,[box],0,(0,0,255),2)
#Decision of large orientation
```

```
    if rotation >= 80 or rotation <= -80:
        print('Too much rotation')
        i=0;
        cv.imwrite('fault_%i.jpg',frame)
        i=i+1

    cv.imshow('procedure',frame)
    cv.imshow('threshold',frame_thresh)


    if cv.waitKey(20) & 0xFF == 27:
        break

cap.release()
cv.destroyAllWindows()
```

B.4 Python Code for Version 2A and 2B

```
import cv2 as cv
import numpy as np

# Load input
cap = cv.VideoCapture('Test.mp4')

#Array to store orientation of each frame
orient_ation = []
count = 0
orient_ation.append(count)

while True:

    #Read input for current frame
    ret1,current_frame = cap.read()

    #Print error message if there is no input
    if ret1 == False:
        print('There is no valid input')
        break

    #Gray-scale conversion of current input
    current_frame_gray = cv.cvtColor(current_frame,cv.COLOR_BGR2GRAY)

    #Comment out to Gaussian blur to simulate version_2B
    #Comment out to median blur to simulate version_2A
   #current_frame_fil = cv.GaussianBlur(current_frame_gray,(25,25),4)
    current_frame_fil = cv.medianBlur(current_frame_gray,19)

    #Thresholding
    ret4,current_frame_thresh = cv.threshold(current_frame_fil,15,255,cv.THRESH_BINARY_INV)

    #Contour Detection
    im2,current_frame_cont,hierarchy =
     cv.findContours(current_frame_thresh,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)

    #Creat array to store detected contours in descending order by their area
    cnt_area_current = sorted(current_frame_cont, key = cv.contourArea, reverse = True)

    #draw contour to original input
    cv.drawContours(current_frame,cnt_area_current[0],-1,(255,0,0),3)
```

```
    #Moments computation
    M_current = cv.moments(cnt_area_current[0])
    cx_current = int(M_current['m10']/M_current['m00']) #CENTER IN X-AXIS
    cy_current = int(M_current['m01']/M_current['m00']) #CENTER IN Y-AXIS

    #Draw center of contour on original input
    cv.circle(current_frame,(cx_current,cy_current),7,(255,0,0),-1)

    #Draw arrow from center of frame to contour center of dark region
    cv.arrowedLine(current_frame,(640,650),(cx_current,cy_current),(0,255,0),10)

    #Index region for direction
    left_index = int((4*current_frame.shape[1])/10)
    right_index = int((6*current_frame.shape[1])/10)

    if cx_current <= left_index:

        cv.putText(current_frame,'Move
Left',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

    elif cx_current >= right_index:

        cv.putText(current_frame,'Move
Right',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

    else:

        cv.putText(current_frame,'Move
Forward',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)


    #Computes rotatable rectangle that fits to contour of dark region
    min_rec = cv.minAreaRect(cnt_area_current[0])
    orient_ation.append(min_rec[2])
    rotation = abs(orient_ation[count]-orient_ation[count+1])
    count = count+1

    #Computes corner points for rotatable rectangle to draw it on original image
    box = cv.boxPoints(min_rec)
    box = np.int0(box)

    #Draw rotatable rectangle to original image
```

```
    cv.drawContours(current_frame,[box],0,(0,0,255),2)

    #Decision of large orientation
    if rotation >= 80 or rotation <= -80:
        print('Too much rotation')
        i=0;
        cv.imwrite('fault_%i.jpg',current_frame)
        i=i+1

    #produce output
    cv.imshow('procedure',current_frame)
    cv.imshow('threshold',current_frame_thresh)

    if cv.waitKey(30) & 0xFF == 27:
        break

cap.release()
cv.destroyAllWindows()
```

B.5 Python Code for Final Version

```
import cv2 as cv
import numpy as np

# Load input
cap = cv.VideoCapture('Test.mp4')

#Create array to store frames causing out of index error
non_contour_frame = []

#Array to store orientation of each frame
orient_ation = []
count = 0
while True:

    ret1,current_frame = cap.read()

    #Print error message if there is no input
    if ret1 == False:
        print('There is no valid input')
        break

    #Gray-scale conversion
    current_frame_gray = cv.cvtColor(current_frame,cv.COLOR_BGR2GRAY)


    #Comment out to Gaussian blur to simulate version_2B_Final
    #Comment out to median blur to simulate version_2A_Final
    #current_frame_fil = cv.GaussianBlur(current_frame_gray,(25,25),4)
    current_frame_fil = cv.medianBlur(current_frame_gray,19)

    #Thresholding
    ret4,current_frame_thresh = cv.threshold(current_frame_fil,15,255,cv.THRESH_BINARY_INV)

    #Contour Detection
    im2,current_frame_cont,hierarchy =.
    cv.findContours(current_frame_thresh,cv.RETR_TREE,cv.CHAIN_APPROX_NONE)

    #Creat an array to store detected contours in descending order by their area
    cnt_area_current = sorted(current_frame_cont, key = cv.contourArea, reverse = True)

    try:
```

```
        #draw contour to original input
        cv.drawContours(current_frame,cnt_area_current[0],-1,(255,0,0),3)

        #Moments computation
        M_current = cv.moments(cnt_area_current[0])

        cx_current = int(M_current['m10']/M_current['m00']) #Center of contour in x-axis
        cy_current = int(M_current['m01']/M_current['m00']) #Center of contour in y-axis

        #Draw center of contour on original input
        cv.circle(current_frame,(cx_current,cy_current),7,(255,0,0),-1)

        #Draw arrow from center of frame to contour center of dark region
        cv.arrowedLine(current_frame,(640,650),(cx_current,cy_current),(0,255,0),10)


        #Index region for direction
        left_index = int((4*current_frame.shape[1])/10)
        right_index = int((6*current_frame.shape[1])/10)

        if cx_current <= left_index:

            cv.putText(current_frame,'Move
Left',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

        elif cx_current >= right_index:


cv.putText(current_frame,'MoveRight',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)

        else:

            cv.putText(current_frame,'Move Forward',(420,100),cv.FONT_HERSHEY_SIMPLEX,2,(0,255,0),5)


        #Computes rotatable rectangle that fits to contour of dark region
        min_rec = cv.minAreaRect(cnt_area_current[0])
        orient_ation.append(min_rec[2])
        rotation = abs(orient_ation[count]-orient_ation[count+1])
        count = count+1
        print('change in orientation of contour: %r'%rotation)
```

```
        #Computes corner points for rotatable rectangle to draw it on original image
        box = cv.boxPoints(min_rec)
        box = np.int0(box)

        #Draw rotatable rectangle to original image
        cv.drawContours(current_frame,[box],0,(0,0,255),2)


        #Decision of large orientation
        if rotation >= 80 or rotation <= -80:
            print('Too much rotation')
            i=0;
            cv.imwrite('fault_%i.jpg',current_frame)
            i=i+1

        #produce output
        cv.imshow('procedure',current_frame)
        cv.imshow('threshold',current_frame_thresh)

        #Pass frame where no contour is detected
    except IndexError:

        #Add frame to array if it causes out of index error due to small contours
        no_contour = cap.get(cv.CAP_PROP_POS_FRAMES)
        non_contour_frame.append(no_contour)
        continue

    #Press ESC to exit
    if cv.waitKey(30) & 0xFF == 27:
        break

#Frame length of the video
length = cap.get(cv.CAP_PROP_FRAME_COUNT)

#Number of frames where out of index error occur
length_none = len(non_contour_frame)

#Calculate error rate of dark region detection
error_rate = ((length-length_none)/length)*100
print(error_rate)

cap.release()
```

```
cv.destroyAllWindows()
```