

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By SWETCHA RAO

Entitled

3D ENDOSCOPY VIDEO GENERATED USING DEPTH INFERENCE: CONVERTING 2D TO 3D

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

LAUREN CHRISTOPHER

Chair

MAHER RIZKALLA

PAUL SALAMA

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): LAUREN CHRISTOPHER

Approved by: BRIAN KING

Head of the Graduate Program

07/20/2012

Date

**PURDUE UNIVERSITY
GRADUATE SCHOOL**

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

3D ENDOSCOPY VIDEO GENERATED USING DEPTH INFERENCE: CONVERTING 2D TO
3D

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22, September 6, 1991, Policy on Integrity in Research*.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

SWETCHA RAO

Printed Name and Signature of Candidate

07/06/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

3D ENDOSCOPY VIDEO GENERATED USING DEPTH INFERENCE:
CONVERTING 2D TO 3D

A Thesis

Submitted to the Faculty

of

Purdue University

by

Swetcha Rao

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2012

Purdue University

Indianapolis, Indiana

To my family.

ACKNOWLEDGMENTS

I express my heart felt thanks and sincere gratitude to Dr. Lauren Christopher, who has been a great guide and support during the course of my thesis. A special thanks to Dr. Aliya Noor for being kind enough to share the bronchoscopy data sets with us and also to Valerie Lim Diemer and Sherrie Tucker for their timely guidance. I would like to thank my friends Ajay Kumar Bandi, Shreya Pandita, and Weixu Li for their encouragement and help, through the duration of my thesis.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vi
LIST OF FIGURES	vii
ABSTRACT	ix
1 INTRODUCTION	1
1.1 Advantages of 3D Endoscopy	1
1.2 Literature Review for 2D to 3D Conversion of Endoscopic Image Sequences	5
1.2.1 2D to 3D Conversion of Endoscopy from Binocular Image Sequence	5
1.2.2 2D to 3D Conversion of Endoscopy from Monocular Image Sequence	7
1.2.3 Preview of Our New Monocular Extensions to the Literature	7
2 DEPTH INFERENCE ALGORITHM FOR ENDOSCOPY	9
2.1 Image Data Preparation	11
2.2 Image Filtering and Canny Edge Detection	12
2.3 Ellipse Detection	13
2.4 Depth-Map Generation	33
3 RESULTS FROM REAL ENDOSCOPY IMAGES: COMPARING OUR RESULTS WITH TRUTH DATA	38
4 CONCLUSION AND FUTURE WORK	50
LIST OF REFERENCES	53
APPENDICES	
A ALGORITHM IMPLEMENTED ON MORE BRONCHOSCOPY IMAGES	55
A.1 Two vanishing points detected from two ellipses	55

	Page
A.2 One vanishing point detected from ellipse and one inferred vanishing point	58
B CODE IN VISUAL C/C++ USING OPENCV LIBRARY	61
B.1 Functions for Stages 1, 2 and 4	61
B.2 Functions for Stage 3: Ellipse Detection	65
B.3 Main Functon	77

LIST OF TABLES

Table	Page
1.1 Summary of Advantages of 3D Endoscopic Surgeries	4
2.1 Synthetic test images and results: testing accuracy of ellipse detection code	22
2.2 Comparison of ellipse parameters of actual and detected ellipses from the synthetic test images in Table 2.1	23
2.3 Synthetic test images and results (continued from Table 2.1): testing accuracy of ellipse detection code with images relevant to endoscopy data	24
2.4 Comparison of ellipse parameters of actual and detected ellipses from the synthetic test images in Table 2.3.	25
2.5 Real-world test images and results	26
2.6 Comparison of ellipse parameters of actual and detected ellipses from the synthetic test images in Table 2.5.	27
3.1 Comparison of ellipse parameters of truth images and inferred depths .	40
3.2 Time elapsed for execution of each stage of our algorithm (seconds) . .	48
4.1 Comparing accuracy of our algorithm with real values	51

LIST OF FIGURES

Figure	Page
2.1 Block diagram of the algorithm	10
2.2 Region of Interest in Endoscopy Images	11
2.3 Effect of masking on edge detection	14
2.4 Example - Image Data Preparation Stage	15
2.5 Ridges of endoscopy image appearing as ellipses in the edge-detected image	16
2.6 Circles detected from OpenCV circle detection code(left) vs our algo- rithm(right)	17
2.7 Parameters for ellipse detection	19
2.8 Code snippet for data storage using accumulator array and accumulator matrix	20
2.9 Code snippet for implementing adaptive voting threshold	20
2.10 Image #1: Example of one vanishing point detected from ellipses . . .	28
2.11 Image #2: Example for two ellipses detected	29
2.12 Image #3: Example of ellipses detected from many edge pixels	30
2.13 Image #4: Example of one ellipse detected for two vanishing points present in source image	31
2.14 Image #5: Example of ellipses detected with bubble clutter in source image	32
2.15 Example of depth gradient generated (right) on tunnel image from circle detection (left)	34
2.16 Depth-map generation example #1: Two ellipses detected	36
2.17 Depth-map generation example #2: One ellipse detected, one inferred .	37
3.1 Truth image (left) and inferred depth with detected ellipses superimposed (right)	39
3.2 Depth gradient obtained for image #1: without surgical instrument . .	42
3.3 Depth gradient obtained for image #2: without surgical instrument . .	43

Figure	Page
3.4 Depth gradient obtained for image #3: without surgical instrument . . .	44
3.5 Depth gradient obtained for image #1: with surgical instrument	45
3.6 Depth gradient obtained for image #2: with surgical instrument	46
3.7 Depth gradient obtained for image #3: with surgical instrument	47
3.8 Graph representing execution time of our algorithm vs number of edge pixels	49
Appendix Figure	
A.1 Result of algorithm implemented on dataset with two detected vanishing points from ellipses	55
A.2 Result of algorithm implemented on dataset with two detected vanishing points from ellipses (continued from Figure A.1)	56
A.3 Result of algorithm implemented on dataset with two detected vanishing points from ellipses (continued from Figure A.2)	57
A.4 Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred	58
A.5 Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred (continued from Figure A.4)	59
A.6 Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred (continued from Figure A.5)	60

ABSTRACT

Rao, Swetcha. M. S. E. C. E. , Purdue University, August 2012. 3D Endoscopy Video Generated Using Depth Inference: Converting 2D to 3D. Major Professor: Lauren Christopher.

A novel algorithm was developed to convert raw 2-dimensional endoscope videos into 3-dimensional view. Minimally invasive surgeries aided with 3D view of the in-vivo site have shown to reduce errors and improve training time compared to those with 2D view. The novelty of this algorithm is that two cues in the images have been used to develop the 3D. Illumination is the first cue used to find the darkest regions in the endoscopy images in order to locate the vanishing point(s). The second cue is the presence of ridge-like structures in the in-vivo images of the endoscopy image sequence. Edge detection is used to map these ridge-like structures into concentric ellipses with their common center at the darkest spot. Then, these two observations are used to infer the depth of the endoscopy videos; which then serves to convert them from 2D to 3D. The processing time is between 21 seconds to 20 minutes for each frame, on a 2.27GHz CPU. The time depends on the number of edge pixels present in the edge-detection image. The accuracy of ellipse detection was measured to be 98.98% to 99.99%. The algorithm was tested on 3 truth images with known ellipse parameters and also on real bronchoscopy image sequences from two surgical procedures. Out of 1020 frames tested in total, 688 frames had single vanishing point while 332 frames had two vanishing points. Our algorithm detected the single vanishing point in 653 of the 688 frames and two vanishing points in 322 of the 332 frames.

1. INTRODUCTION

Minimally Invasive Surgery (MIS) is a surgical technique which, when compared to open surgeries, has advantages like reduced scarring and shorter recovery time. For these reasons, MIS is replacing other standard surgical procedures. The procedure for MIS requires a long and thin tube with a miniature camera attached at its end. Such a tube is called an Endoscope, as it is used to view internal structures in the body. This enables surgeons to view the surgical area in a screen from the miniature camera. This research converts the 2D video output from such a camera, making it viewable on 3D-Television. Viewing in 3D has shown to improve clinical practice by speeding up execution time of the surgical procedure, improving the quality of training for novice practitioners and ultimately yielding better patient outcomes.

The following sections of this chapter explain the need and advantages of having 3D endoscopy compared to 2D, and the existing methods, algorithms and research on conversion of 2D endoscopic vision to 3D.

1.1 Advantages of 3D Endoscopy

With the increasing use of minimally invasive surgeries, the ability to view the endoscopy by practitioners and resident surgeons in 3D is an added advantage. The statistical evidence from a number of studies prove that 3-dimensional view of the in-vivo surgical site improves time for learning endoscopic procedures and also reduces the operating time among practitioners. Studies also show that compared to 2D, 3D endoscopy provides a more realistic view of the surgical site, thus making it advantageous over 2D endoscopy when used during delicate endoscopic surgeries. The following paragraphs cite and discuss some studies conducted by researchers to

analyze the effects of endoscopic surgeries performed using 3D vision as compared to 2D vision.

In a recent presentation at the American Rhinologic Society COSM (Combined Otolaryngology Society Meetings), a study was conducted to analyze the effects of 3D endoscopic surgery compared to using a 2D system [1]. The experiment involved 7 patients undergoing sinonasal and skull base surgeries. On some of these patients, a 2D endoscopic system was used to perform these surgeries while on the others, 3D endoscopic system was incorporated at key portions of the procedure. The reported result is that by 3D endoscopy, the depth perception and endoscopic orientation were enhanced in 43% of the patients, without any increase in complications due to the surgery in the patients.

In an article written by *Taffinder. N*, et al [2], twelve experienced laparoscopic surgeons and sixteen novices were asked to perform a 672 tasks of laparoscopic surgeries: comparing 2D, 3D, and direct vision using the Imperial College Surgical Assessment Device (ICSAD); standard objective scores for measuring the movements of surgical instruments. Compared with direct vision, the 2D endoscopic vision reduced the performance by 35%-100%, while the 3D reduced these mistakes by 41%-53% in both novices and experienced surgeons. Also, no side effects were reported in the new 3D system.

J. C. Byrn, et al, in their journal paper [3], talk about a study designed to evaluate the effect of 3-dimensional vision on the performance of experienced and resident surgeons compared to 2-dimensional vision, using the da Vinci surgical system. In this study, the performance times and errors of six surgeons and six senior surgical residents were recorded using 2D and 3D vision individually for various tasks. The four tasks or skills that were tested were bead transfer and drop, needle capping, threading and knot tying. The paper reports that statistical calculations of error rates and performance times for all the 4 skills were reduced by 34% to 46% and 44% to 66% respectively. This proves that 3-dimensional vision has a significant

advantage compared to 2D vision in the improvement of performance and error rate in both experienced and novice surgeons.

Apart from endoscopy, enhanced depth perception in laparoscopy with 3D imaging has been reported as a major advantage of minimally invasive surgical procedures [4]. In this the participants included 20 novices, 20 practitioners with an experience of 50 laparoscopic procedures and 20 with an experience of more than 50 laparoscopic procedures. Results show clearly that there was a significant improvement in speed and accuracy under 3 dimensional conditions compared to 2D vision on all levels of experience, irrespective of the order of the sequence of each individual test. The performance time decreased by 21.6-27.2% and number of mistakes reduced by 24.6-80.4% in the 3D assisted surgical procedures. Conclusion of this evaluation states that "three dimensional imaging may further improve the safety aspect of MIS." Table 1.1 summarizes the conclusions from the above cited references.

For minimally invasive surgical methods, apart from improved accuracy, speed, dexterity and safety offered by 3-dimensional vision, another feature that adds to its advantage is a new MIS system under development: Telepresence surgery [5]. This technology uses remote force-feedback manipulators, 3D vision and stereophonic sound, projecting the images from the site of surgery to a remote workstation, allowing the surgeon to perform the surgery by not being present at the surgical site [6].

The above 4 studies are representative of other multiple studies where researchers show that 3-dimensional endoscopic view is advantageous over 2-dimensional view in terms of performance, error minimization and reduced learning curve. In this section, the advantages of 3D endoscopy have been discussed, the next section discusses about existing methods to convert 2D endoscopy to 3D, and in what aspects the method developed in this thesis is advantageous over the prevailing methods.

Table 1.1
Summary of Advantages of 3D Endoscopic Surgeries

Reference Article	# of patients/tasks	Conclusions
[1]	7 patients	Enhanced depth perception and endoscopic orientation by 3D endoscopy. Improved effect of 3D endoscopic surgery in 43% of patients.
[2]	672 tasks	Compared to direct vision performance based on movement of surgical instruments reduced by 35-100% in 2D surgical vision and these mistakes decreased by 41-53% in 3D surgical vision.
[3]	48 tasks	Compared to 2D vision, 3D vision reduced task execution time by 34-46% Error rate by 44-66%
[4]	120 tasks	Compared to 2D, 3D vision reduced the task execution time and mistakes by 21.6-27.2% and 24.6-80.4% respectively.

1.2 Literature Review for 2D to 3D Conversion of Endoscopic Image Sequences

There are several algorithms to estimate the depth associated with 2D images, to reconstruct its 3D view. According to the depth cues on which the algorithms rely, they are classified into 12 categories [7]. In terms of 3D endoscopy, many researchers and computer vision enthusiasts have developed methods for to convert 2D endoscopy to 3D view based on stereoscopic depth-map generation. Broadly, there are two methods adopted to convert 2D endoscopy to 3D. They are:

- (1) 3D from binocular image sequence, and
- (2) 3D from monocular image sequence.

In this thesis, 3D is developed from the monocular image sequence. The following sub-sections consist of literature review of 2D to 3D conversion of endoscopic image sequences using each of the above methods.

1.2.1 2D to 3D Conversion of Endoscopy from Binocular Image Sequence

Many of the methods used by researchers associate binocular vision to the endoscope, which means that to develop 3D endoscopy, the hardware is modified to consist of two cameras.

K. Keller and A. State, of InnerOptic Technology Inc., have developed a stereo endoscopy hardware system, that plugs in to the existing monoscopes already owned by a majority of practitioners [8]. This hardware consists of 5.5mm and 10mm dual optical channel laparoscopes, which combine both exit channels of the laparoscope into a single, standard, endoscopic eye cup. This technology helps practitioners to reduce the cost of buying new hardware, compatible with 3D technology. Although, practitioners still have to invest on buying the stereo endoscopy hardware to plug in to their monoscopes, which can be avoided by adopting our proposed method of converting 2D endoscopy to 3D.

M. Chan, et al, designed a system that consists of two imaging channels and a provision to illuminate the imaged object, to measure the surface topology [9]. The imaging system designed, consisted of two wide-angle lenses with each imaging channel measuring 1.8mm in diameter. According to this model, the total diameter of the endoscope including the stereo cameras, stainless steel tubing and separation between the lenses sums up to 5.1mm. While it is known that the minimum diameter of existing single channel endoscopes manufactured to date is 5.0mm, the drawback of this design is the slightly increased size of the endoscope [9]. Although the difference in diameters is not much, as diameter of Bronchioles range from 0.8mm to 8.0mm, it is always desirable to minimize the diameter of the endoscope.

As another example, a computerized 3-dimensional endoscopic imaging system was designed, dedicated to delicate endoscopic surgery [10]. In this design, a 3D telescope was used to capture stereo images of the endoscope. This telescope consisted of a stereo laparoscope of diameter 10.0mm, attached to a camera hand-piece containing 2 microchip cameras. Although in this design, high resolution cameras were used to obtain the images, and the results conclude that the objective: reducing surgical performance time surgeons surgeons was achieved. However, the size of the endoscope (10.0mm) is still too big, which restricts this design from being applied to bronchoscopes which demand the outer diameters to be in the range of 2.2mm to 4.9mm in case of pediatric sized flexible bronchoscopes [11].

In a publication by F. Mourgues, F. Devemay, and E. Coste-Maniere [12], a stereoscopic endoscope was used to obtain the pre-operative images for surgeons to view the operative field in 3D. The novelty of this paper is that instruments are removed from the field of view in the 3D version, so that surgeons can get a view of the internal organs as if they were looking at them directly. Although this algorithm gives a more realistic view of the surgical site, there were discrepancies in this algorithm due to correlation errors in the stereoscopy algorithm, resulting in errors in proper reconstruction of 3D perspective.

1.2.2 2D to 3D Conversion of Endoscopy from Monocular Image Sequence

In terms of depth-map estimation of 2D endoscopic image sequences to convert them to 3-dimensional view from monocular input image sequence, most of the previous attempts made by researchers consist of structure-from-motion algorithms.

In [13], a structure-from-motion algorithm was used for a sparse set of feature points and a fast, linear interpretation algorithm was developed for creating a dense disparity field for synthesizing stereoscopic views from original monocular video. This algorithm uses the monocular image sequence to interpret the depth of the endoscopic video by using the normalized depth of every pixel and then recovering the normalized depth via structure-from-motion and linear interpolation. As reported by the authors, the results of this algorithm demonstrate feasibility and effectiveness. However, there are inconsistencies in the dense disparity map for real endoscope data.

In the algorithm developed in [14], reconstruction of the 3D scene of endoscopic image sequence from monocular endoscopy is done by estimating the motion of the camera, then generating a triangle mesh and calculating 3D coordinates based on the triangle mesh and motion of the camera. Then, to assign a natural look to the 3D reconstruction, final texturing is done. This approach is robust and could have a good potential after the improvement of its accuracy and after analysis of the estimation error.

1.2.3 Preview of Our New Monocular Extensions to the Literature

In this thesis, we use two novel cues to estimate and infer the depth of endoscopic images, from monocular endoscopes, which are as follows:

- (1) Depth from illumination or shadowing.
- (2) Depth from ellipses and vanishing point.

The implementation of our algorithm is shown on synthetic truth data and on real

bronchoscope data and the results are discussed based on mean square error calculations and number of proper vanishing points detected among all the frames tested.

This thesis is structured as follows: In Chapter 2, the implementation of the algorithm is described stage-wise, starting with the data preparation stage to the depth-map generation and depth image based rendering (DIBR). The results of the algorithm, implemented using *Open Source Computer Vision (OpenCV)* are shown in Chapter 3, with real as well as synthetic images given as inputs. Chapter 4 concludes and elucidates on the challenges faced during the algorithm development and implementation, and suggests future work, to improve the algorithm.

2. DEPTH INFERENCE ALGORITHM FOR ENDOSCOPY

This chapter explains the algorithm developed in this thesis for conversion of monocular 2D endoscopy video into 3D and its implementation. Two cues are used to develop depth-map for monocular endoscopic image sequences. The first cue is that the edge detected images of endoscopy are characterized by ridge-like structures that appear concentric, with their centers at the darkest point in the source image where light from the camera cannot reach. This dark point is the second cue used in this algorithm. The results of this researched were tested on two data sets, obtained from Dr. Aliya Noor at Indiana University (IU) Hospital, Pulmonology Department.

The proposed algorithm for depth inference consists of four stages. They are as follows:

- (1) Slice the endoscopy video into a stack of images and select the region of interest.
- (2) Filter the Canny edge detected image in the stack and apply a novel combination of image processing techniques.
- (3) Detect ellipses from the edge information of each image.
- (4) An atypical approach to generate depth-map from the ellipse parameters obtained.

These stages are described in a block diagram in Figure 2.1. In Bronchoscopy videos there are often two branches called Bronchioles; which when seen from an image-processing perspective appear to have two vanishing points. In such cases, the algorithm detects two sets of concentric ellipses at each of the vanishing points. Once the depth-map is generated, it is stitched to the right side of the source image for Depth Image Based Rendering. This is repeated for each image in the stack and upon completing with the entire stack, it is converted back into a video sequence. This is done in the last stage of the algorithm. Each stage is further explained in the following sections.

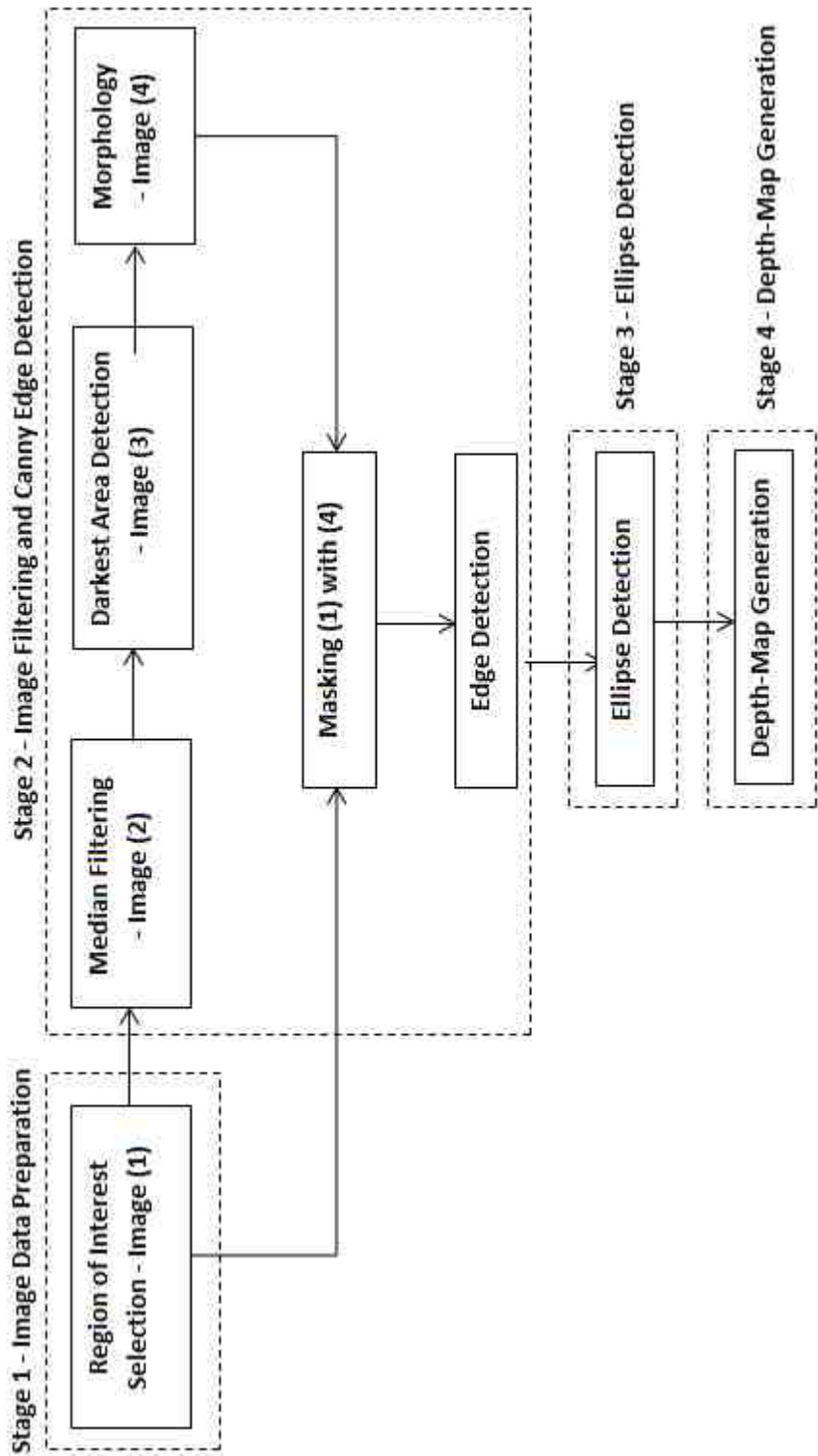


Fig. 2.1. Block diagram of the algorithm

2.1 Image Data Preparation

The endoscopy video is first converted from the original movie into a stack of single images at 30 frames per second. Each of these images is taken as a single source image during the process of depth estimation and the algorithm is designed to work on one image at a time.

Usually, an endoscopy video consists of region around the endoscope cameras display, with information about the patient, time and date when the endoscopy was performed, etc. The text is embedded in the endoscopy video over a black background. As our region of interest is the area consisting of only the internal organs, in this stage just the area with the subject of interest is selected and saved. The dimensions of region of interest can be different for different datasets, as they are taken from different endoscopes. This can be a user set parameter for different input endoscope images. This is explained in Figure 2.2, where the inner rectangle is the in vivo surgical site and our region of interest and the area between inner and outer rectangles is where the patients details are embedded.

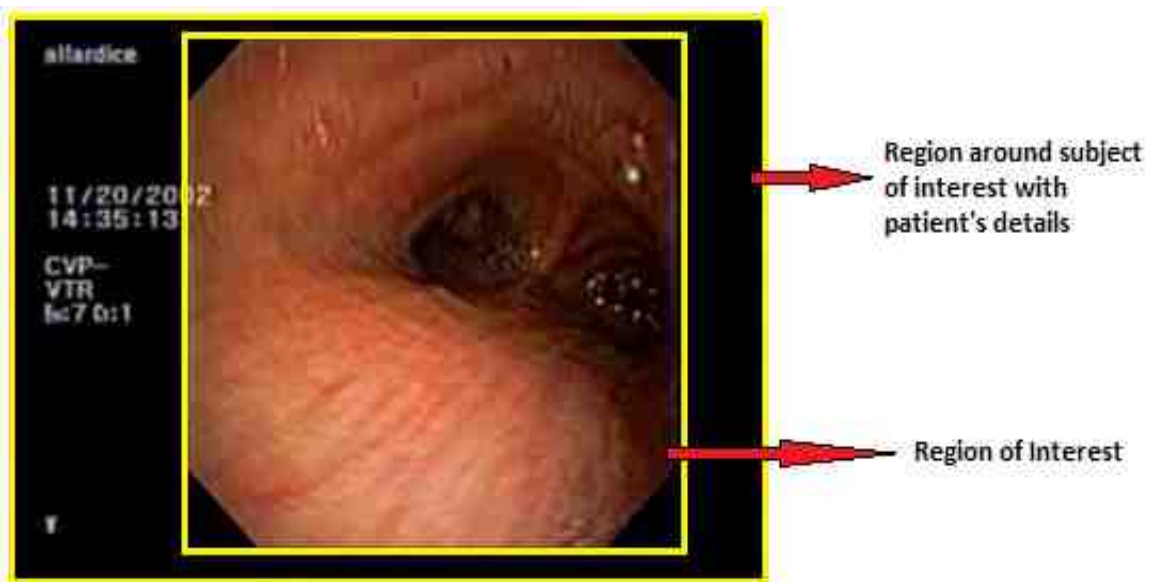


Fig. 2.2. Region of Interest in Endoscopy Images

For the endoscope data set obtained from Dr. Aliya Noor, the dimensions of the rectangular region of interest are 226×213 . These dimensions can vary with the type of camera used in different endoscopes. Further processing is done on this new image, named *roiimg*. Once the depth-map generation step is done, the *roiimg* and depth image are fitted back with the rest of the black part with patients information, so that image dimensions of the source and depth images agree for the final stage (Figure 2.1).

2.2 Image Filtering and Canny Edge Detection

The aim of this stage is to perform edge detection on the output image from the previous stage (Section 2.1), to detect ellipses close to the vanishing point(s) of the endoscopy video. An example of this stage is as shown in Figure 2.3.

First, we pre-process the image from the previous stage (Section 2.1) by applying Median Filter with a square aperture of size 5×5 . This process filters out the specular points appearing in the *roiimg* image due to the reflections from the endoscope light. The median filtered image is referred to as Image 2 (with respect to block diagram in Figure 2.1). To reduce undesired image clutter that appears as noise to the edge detector, further processing is necessary. For this reason, a mask is created by performing morphological operation of *dilation* on the image obtained after darkest area detection on the median filtered image.

To create the mask, the next step is detecting the darkest area. The regions in the endoscopy image where minimum light from the endoscope reaches the field of view are considered as the deepest points for the depth-map generation. The mask is transparent in the darkest areas detected and opaque in the remaining portions of the image. This image is referred to as Image 3 in Figure 2.1. The darkest area in Image 2 is identified by setting a threshold of *95*, *50* and *65* for the red, green and blue components respectively, of the median filtered image. These values are worked

for various in-vivo surgical images. The threshold on the red component is set higher compared to blue and green components since red is a dominant tissue color.

From observation and many trials, it is decided that the most suitable mask is obtained by performing two iterations of dilation on the darkest area detected image (Image 3 in Figure 2.4), with a circular kernel of size 3x3. This double dilated image is now used as a mask over the median-filtered image, to mask out regions in the median-filtered image that cause undesired edge pixels upon edge-detection (Figure 2.3). The resulting masked image from the above step is segmented using *Canny Edge Detection* [3]. The edge pixels are used to detect ellipses in the endoscopy images for depth-map estimation, in the later stages. The lower threshold used for Canny edge detection is 70 and the upper threshold is 100. The aperture size for the Sobel operator used in the Canny edge detection algorithm is 3x3. Figure 2.4 shows this process with an example. The edge detected image is then used in the ellipse detection stage, described in the next section.

2.3 Ellipse Detection

The next stage is ellipse detection. This is one of the novel parts of the algorithm developed in this thesis. A review of classical methods is discussed in this section, followed by the advantages of adopting ellipse detection compared to the classical methods.

When seen from an edge detection perspective, one of the observations made during this research was that the field of interest of the videos and image sequences from Bronchoscopy and Colonoscopy were characterized by elliptical ridge-like structures. These ellipses were complete and closed in some frames, while they were broken in most of the frames. Moreover, the elliptical structures were observed to be nearly concentric, with the centers of the ellipses within some threshold (Figure 2.5).

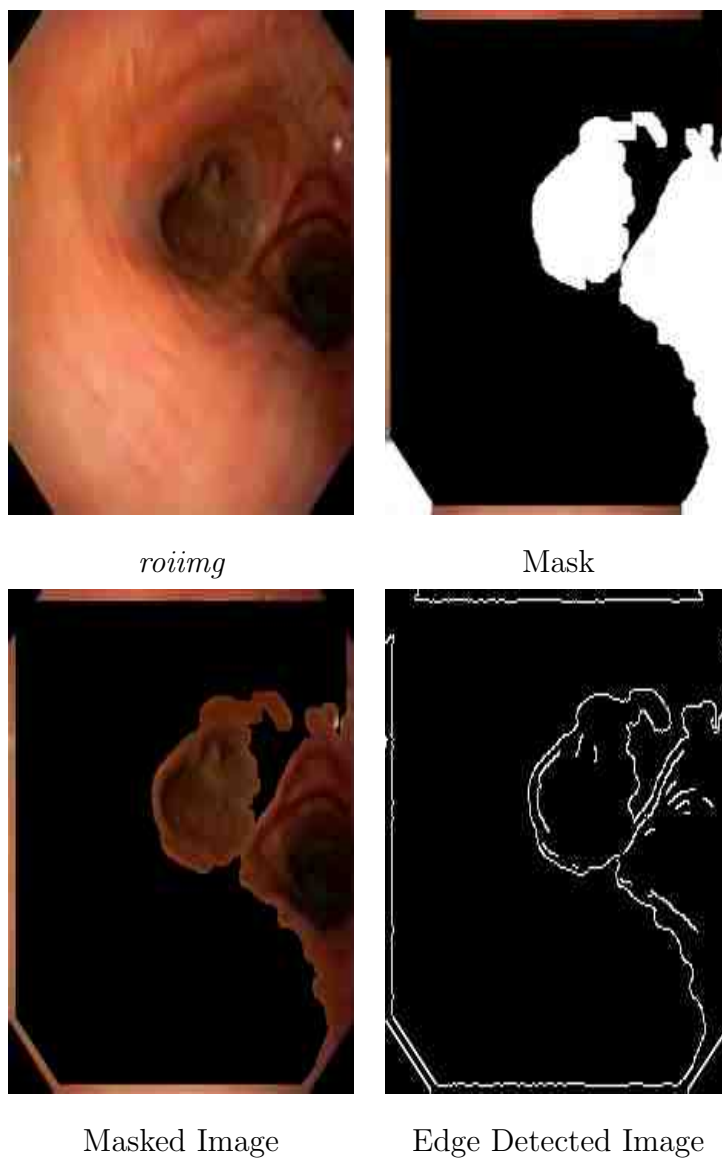


Fig. 2.3. Effect of masking on edge detection

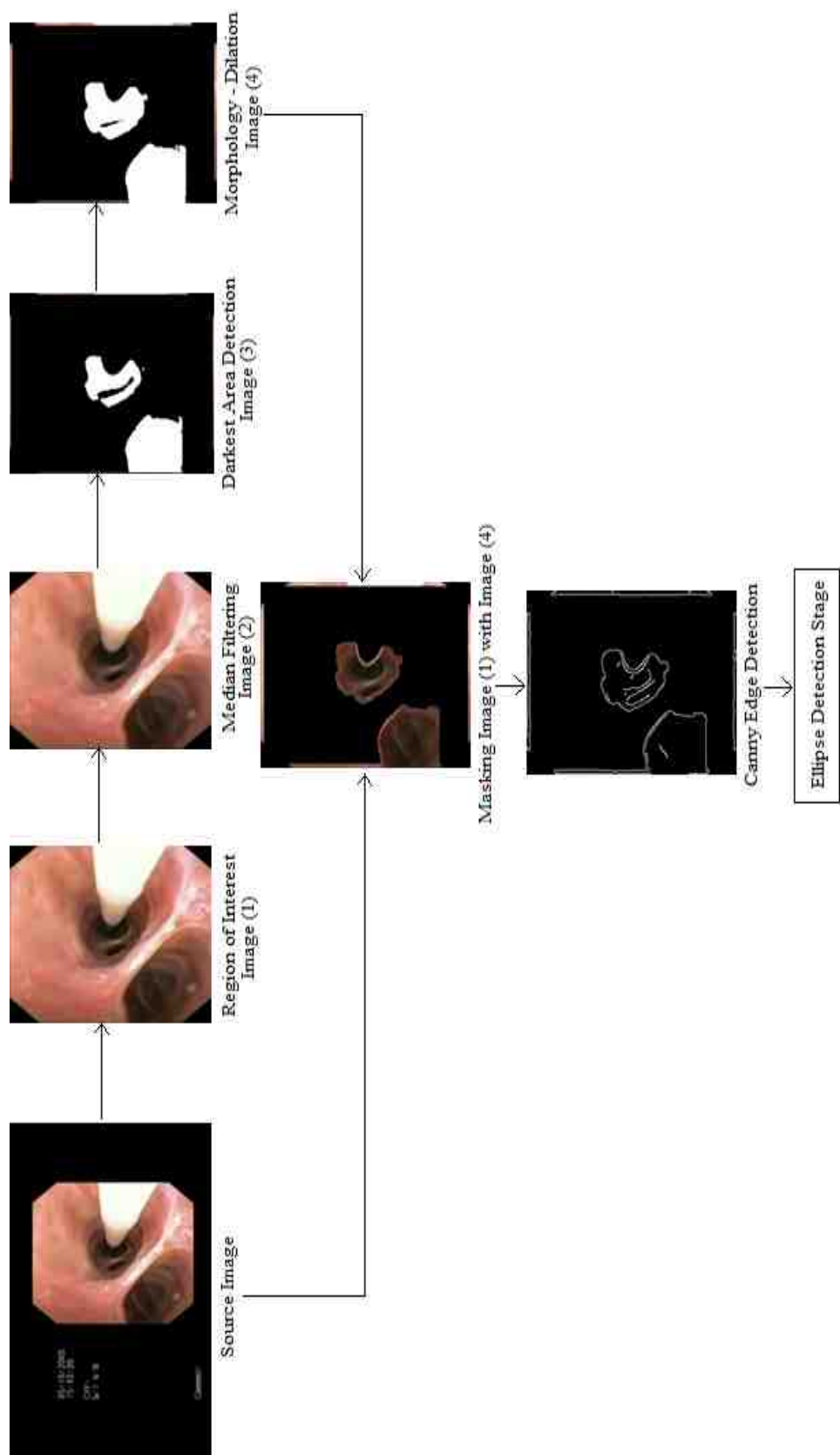


Fig. 2.4. Example - Image Data Preparation Stage

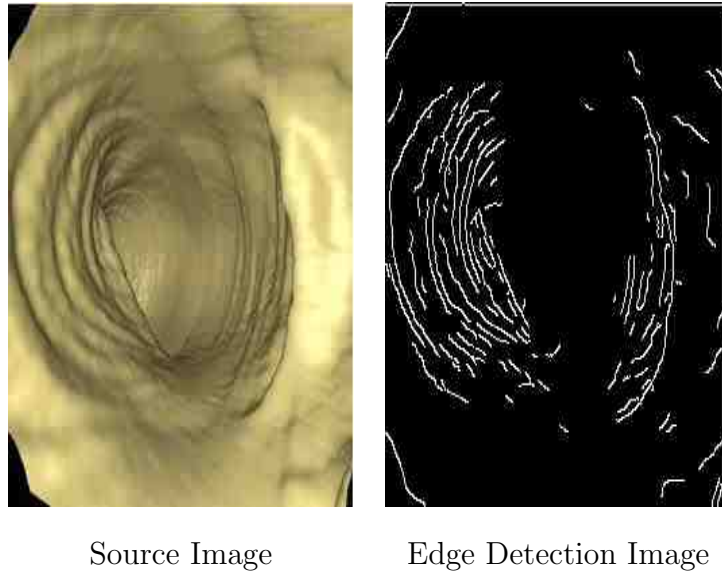


Fig. 2.5. Ridges of endoscopy image appearing as ellipses in the edge-detected image

On the basis of these observations, our research focus was to automatically detect the elliptical shapes appearing in these edge-detected images and then to estimate the depth of each frame of the endoscope image sequence.

We first explored using the OpenCV function for detecting circles. This algorithm for circle detection is based on Hough's method of line detection [15], but was found to eliminate concentric circles as outliers. Therefore, changes to this algorithm were necessary to keep circles having centers close to each other. However, it was observed that the number of outliers increased as the algorithm considers undesired edge pixels as circles. Figure 2.6 shows a comparison of circle detection using OpenCV and our algorithm.

D. H. Ballard, in his paper on generalized Hough transform to detect arbitrary patterns [16], generalizes the Hough algorithm [15] to use edge information to define mapping between the orientations of edge pixels from a reference point. This paper illustrates the implementation of their algorithm by finding the parameters of an ellipse. On similar grounds, detection of ellipses by finding the parameters of possible

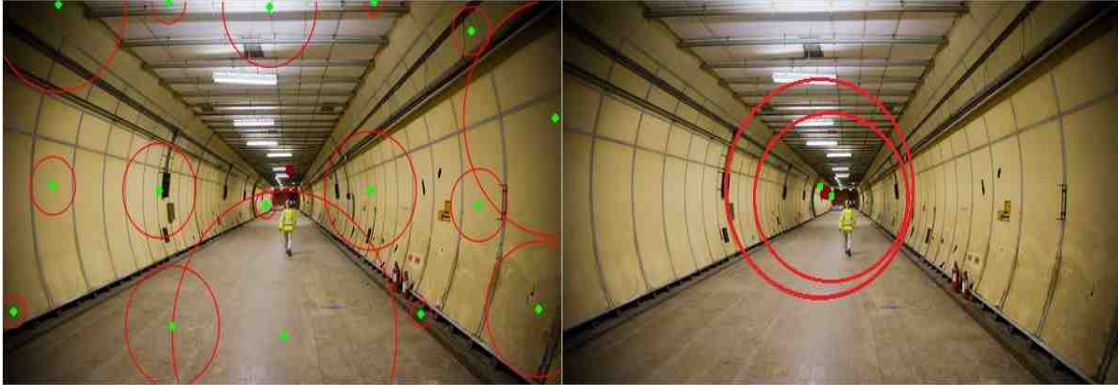


Fig. 2.6. Circles detected from OpenCV circle detection code(left) vs our algorithm(right)

ellipse and voting for the best fitting ellipse was proposed by *Yonghong Xie and Qiang Ji*, in their conference paper [17].

Upon researching a number of journals and conference proceedings, we find that although many researchers have attempted to detect ellipses automatically, the computational complexity of detecting ellipses is high due to the requirement to find all the five parameters of the ellipses. While detecting single ellipses in an image is not very difficult, detection of multiple ellipses is highly complex. Hence, to make sure that the computational complexity of detecting ellipses is as low as possible, we have implemented the algorithm in [17], with some modifications.

According to *Yonghong Xie and Qiang Ji* [17], the edge image is given as input to the ellipse detection function. The white pixels in the edge image represent the image pixels to be considered for ellipse detection. The following describes our algorithm for ellipse detection [17] along with snippets of the code developed using Visual Studio and OpenCV libraries.

The first step in [17] is to store all the edge pixels in a 1 dimensional array. We modify this step by storing the edge pixels in a 2 dimensional array of size $width \times height$ of the edge image, to speed up the process of searching through each edge pixel. The step (3) of [17] where for each pixel (x_1, y_1) and each other pixel (x_2, y_2) of the edge image, the distance between (x_1, y_1) and (x_2, y_2) is calculated, we modify this step by

setting limits to the coordinates of the edge pixels around the borders of the image. This modification reduces the execution time taken by the ellipse detection algorithm as the number of edge pixels that are to be searched to find an ellipse is reduced. Now, if the distance between each edge pixel, (x_1, y_1) and each other edge pixel, (x_2, y_2) is greater than or equal to 8, then the steps (5) through (7) from [17] are carried out.

(5) For the pair of pixels (x_1, y_1) and (x_2, y_2) , using Equations 2.1 to 2.4 to calculate the center, orientation and length of major axis for the assumed ellipse.

(6) For each third pixel (x, y) , if the distance between (x, y) and (x_2, y_2) is greater than the required least distance for a pair of pixels to be considered then carry out the following step.

(7) Using Equations 2.5 and 2.6 to calculate the length of minor axis.

$$x_0 = (x_1 + x_2)/2 \quad (2.1)$$

$$y_0 = (y_1 + y_2)/2 \quad (2.2)$$

$$a = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}/2 \quad (2.3)$$

$$\alpha = \arctan [(y_2 - y_1)/(x_2 - x_1)] \quad (2.4)$$

$$b^2 = (a^2 d^2 \sin^2 \tau)/(a^2 - d^2 \cos^2 \tau) \quad (2.5)$$

where $\cos \tau$ is

$$\cos \tau = (a^2 + d^2 - f^2)/(2ad) \quad (2.6)$$

where ' d ' is the distance between (x, y) and (x_0, y_0) and ' f ' is the distance between (x, y) and (x_2, y_2) . The ellipse parameters are as shown in Figure 2.7.

Steps (8), (9) and (10) of [17] are modified to decrease the execution time of the ellipse detection algorithm. According to step (8) in Y. Xie and Q. Ji's algorithm, after calculating the length of the minor axis, the accumulator array is incremented by 1 for this length of minor axis. Our modification to this step is to consider only those values of 'b' calculated from Equations 2.5 and 2.6, that are greater than a threshold of '8'. This way, ellipses detected from the edge pixels coming up due

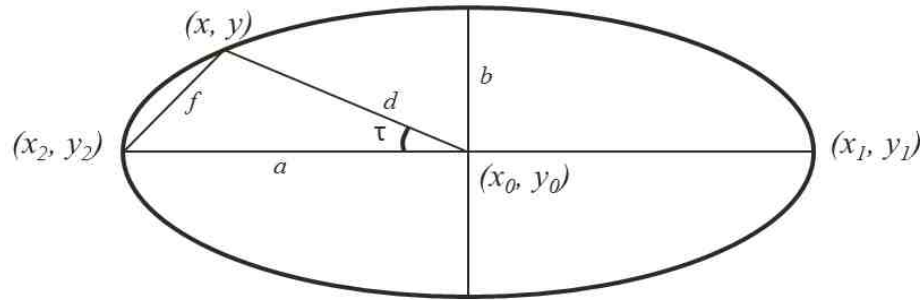


Fig. 2.7. Parameters for ellipse detection

to the bubble clutter in the in-vivo site are eliminated. The threshold is chosen as '8' from observation of a number of bronchoscopy video frames. Next, the values of semi-minor axis that are greater than '8' are stored in a one dimensional array, $ac[]$. The edge pixels corresponding to these values of semi-minor axis are stored in an accumulator matrix, $accum$ of data type $CvMat$. This is repeated until all the pixels are computed for this pair of pixels. A snippet of the code for this part of the algorithm is shown in Figure 2.8.

In step (10), instead of setting a general voting threshold for all types of input images as according to [17], a conditional voting threshold is used. If the total number of edge pixels is less than '1600', then an ellipse is said to be detected if the accumulator vote for the semi-minor axis length is between '27' and '30'; else, an ellipse is detected if the accumulator vote for the semi-minor axis length is greater than '47'. The reason for making this modification is that when a general voting threshold is set, in cases where the number of edge pixels is less and the voting threshold is too high, the correct ellipses are discarded by the algorithm due to insufficient votes; whereas when the number of edge pixels is more and voting threshold that is set, is a low value, then the algorithm ends up detecting too many outliers. Hence, an adaptive voting threshold that changes with the number of edge pixels is a good way to reduce the possibility of erroneous ellipse detection. A snippet of the code for this part of the algorithm is shown in Figure 2.9.

```

/* if semi-minor axis length is greater than '8' */
if(cvRound(b)>8)
{
    /* store the location of edge pixel corresponding to */
    /* this (greater than '8') value of semi-minor axis */
    /* in accumulator matrix */
    ((int*)(accum->data.ptr + accum->step*y))[x] = cvRound(b);
    /* increment accumulator array for this value of semi-minor axis */
    ac[count] = b;
    count++;
}

```

Fig. 2.8. Code snippet for data storage using accumulator array and accumulator matrix

```

/* setting voting threshold according to the total number of */
/* edge pixels in the input edge detected image */
if(((tot<=1600)&&(vote > 27)&&(vote<30)) || ((tot>1600)&&(vote>47)))
{
    /* carry out steps (11) through (15) from*/
    /* Y. Xie and Q. Ji algorithm */
}

```

Fig. 2.9. Code snippet for implementing adaptive voting threshold

- Next, steps (11) through (15) from [17] are implemented. The steps are as follows:
- (11) Output ellipse parameters.
 - (12) Remove the pixels on the detected ellipse from edge pixel array.
 - (13) Clear accumulator array.
 - (14) Loop until all pairs of pixels are computed.
 - (15) Superimpose detected ellipses on the original image.

This algorithm implementation is first tested on synthetic and real world images consisting of simple and slightly complex ellipses, before implementing it on the endoscopy image sequences. The synthetic source images and ellipses detected are shown in Tables 2.1 and 2.3. Table 2.1 consists of synthetic ellipses that are used to test the accuracy of the code implemented for ellipse detection. Table 2.3 consists of ellipses synthesized to test on images that are close to the edge-detected images of endoscopy videos. Table 2.5 shows ellipses detected when the source images are those of real-world scenes. Tables 2.2, 2.4 and 2.6 compare the results of detected ellipse parameters with the actual ellipse parameters, for the synthetic and real world test images.

The parameter values are detected with accuracy of 98.98% to 99.99%. The percentage accuracy, A measured is calculated using Equation 2.7.

$$A\% = 100 - \frac{\sum_{p=1}^5 \left| \frac{detected_p - actual_p}{actual_p} \right|}{5} \quad (2.7)$$

where $detected_p$ and $actual_p$ are the values of the p^{th} detected and actual parameters of the ellipse, respectively.

Next, with these tests concluding the effectiveness of the ellipse detection algorithm and its code, we apply this ellipse detection to the endoscopy images to find the centers and orientations of the elliptical ridges present. Figures 2.10 to 2.14 show some of the bronchoscope image sequences obtained from Dr. Aliya Noor and the ellipses detected.

Table 2.1
Synthetic test images and results: testing accuracy of ellipse detection code

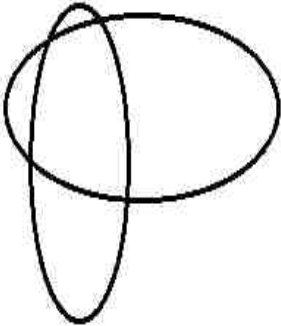
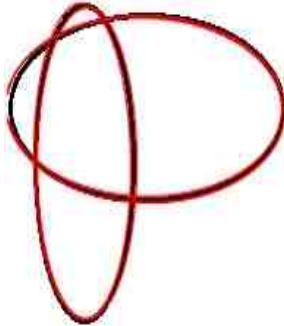
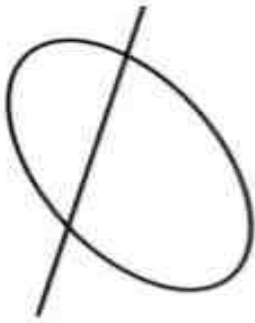

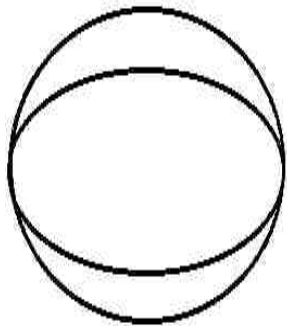

Name of Image	Source Image	Ellipses Detected (Superimposed on Source Image In Red)
Oval 1		
Oval 2		
Oval 3		

Table 2.2
 Comparison of ellipse parameters of actual and detected ellipses from
 the synthetic test images in Table 2.1

Image		x_o	y_o	Orientation Angle	Major Axis	Minor Axis	Accuracy %
Oval 1	Actual	93	47	0	93	46	99.97
		34	82	90	83	34	99.97
	Detected	96	48	0	92	48	
		36	83	89	83	36	
Oval 2	Actual	62	92	56	79	42	99.96
	Detected	60	93	62	80	43	
Oval 3	Actual	126	83	0	126	50	99.98
		130	79	0	127	75	99.98
	Detected	128	80	1	129	51	
		128	81	0	129	76	
		116	65	-4	117	62	

Table 2.3
 Synthetic test images and results (continued from Table 2.1): testing accuracy of ellipse detection code with images relevant to endoscopy data

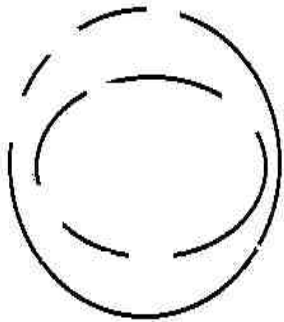

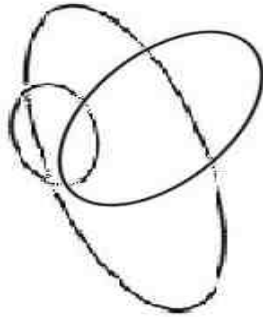

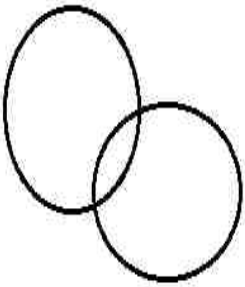

Name of Image	Source Image	Ellipses Detected (Superimposed on Source Image In Red)
Oval 4		
Oval 5		
Oval 6		

Table 2.4
 Comparison of ellipse parameters of actual and detected ellipses from
 the synthetic test images in Table 2.3.

Image		x_o	y_o	Orientation Angle	Major Axis	Minor Axis	Accuracy %
Oval 4	Actual	117	104	0	90.5	55.5	99.95
		118	103	0	106.5	92.5	99.98
	Detected	117	103	0	88	57	
		117	101	0	108	94	
Oval 5	Actual	97	101	-55	78	44	99.95
		79	143	73	129	45	99.98
		40	126	85	39	23	99.95
	Detected	100	113	-55	81	45	
		80	140	75	128	46	
		39	126	89	41	25	
Oval 6	Actual	85	71	0	53.5	51	99.98
		159	111	0	57.5	43.5	99.98
	Detected	85	72	0	53	48	
		155	110	0	58	45	

Table 2.5
Real-world test images and results

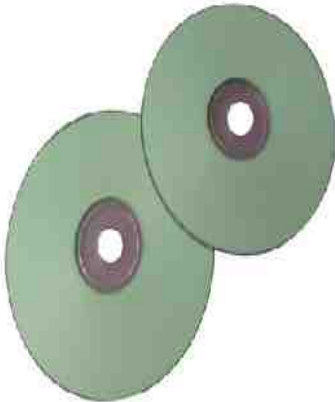
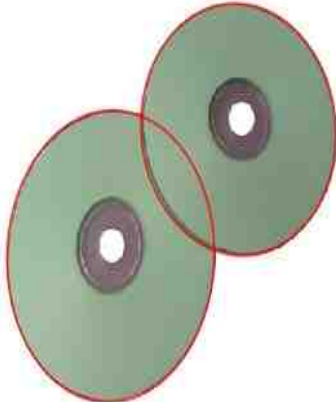




Name of Image	Source Image	Ellipses Detected (Superimposed on Source Image In Red/Blue)
Oval 7		
Oval 8		
Oval 9		

Table 2.6
Comparison of ellipse parameters of actual and detected ellipses from
the synthetic test images in Table 2.5.

Image		x_0	y_0	Orientation Angle	Major Axis	Minor Axis	Accuracy %
Oval 7	Actual	87	150	-87	83	80.5	99.99
		185	77	-2	75	72	99.89
	Detected	87	150	-88	83	81	
		185	77	-3	77	72	
Oval 8	Actual	144	63	0	73	62	99.97
	Detected	145	64	0	81	62	
Oval 9	Actual	157	156	0	150	149	98.99
		116	98	-90	42	15	99.98
		197	98	-90	42	14.5	99.96
	Detected	155	156	-6	150	149	
		116	97	-87	43	15	
		195	97	-87	43	16	

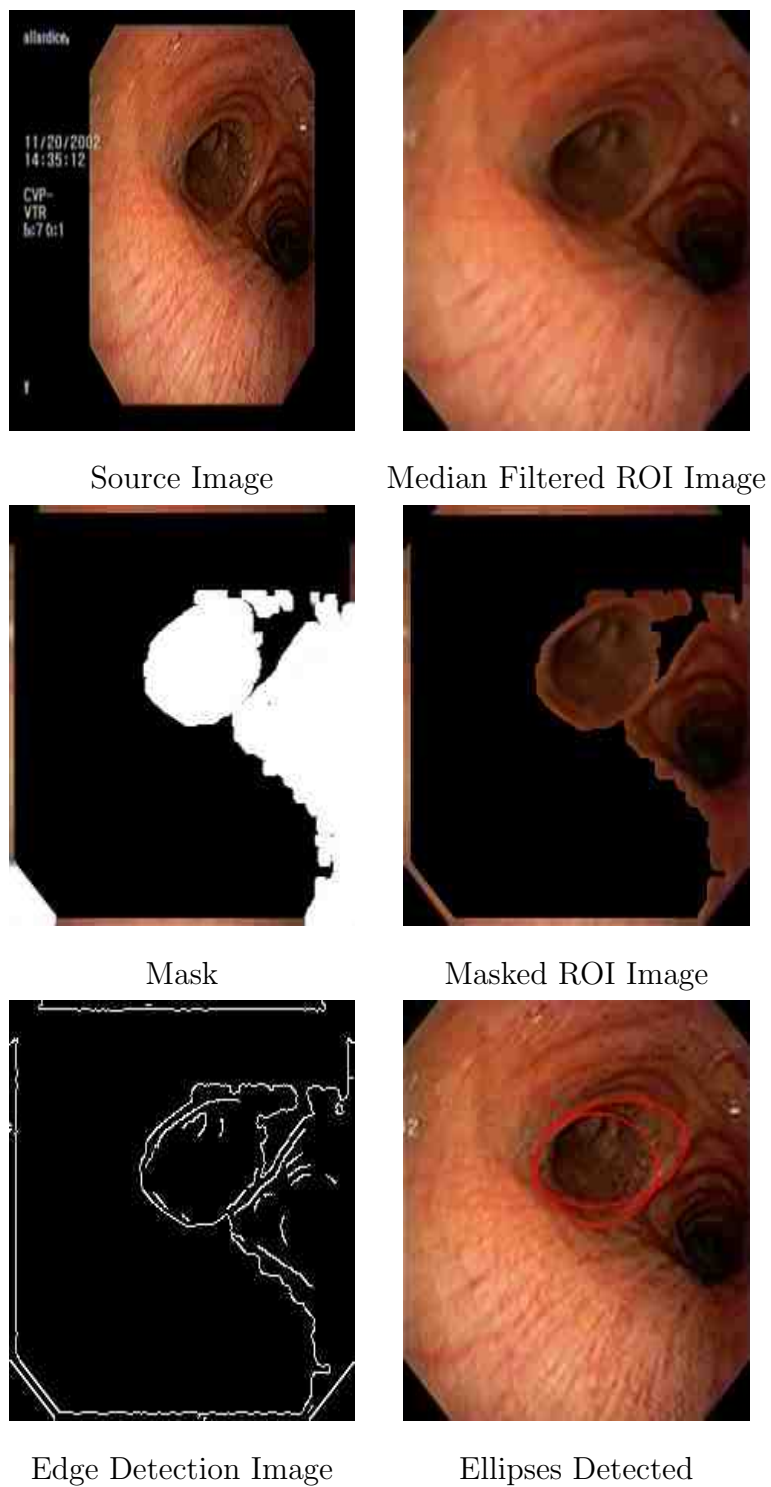


Fig. 2.10. Image #1: Example of one vanishing point detected from ellipses

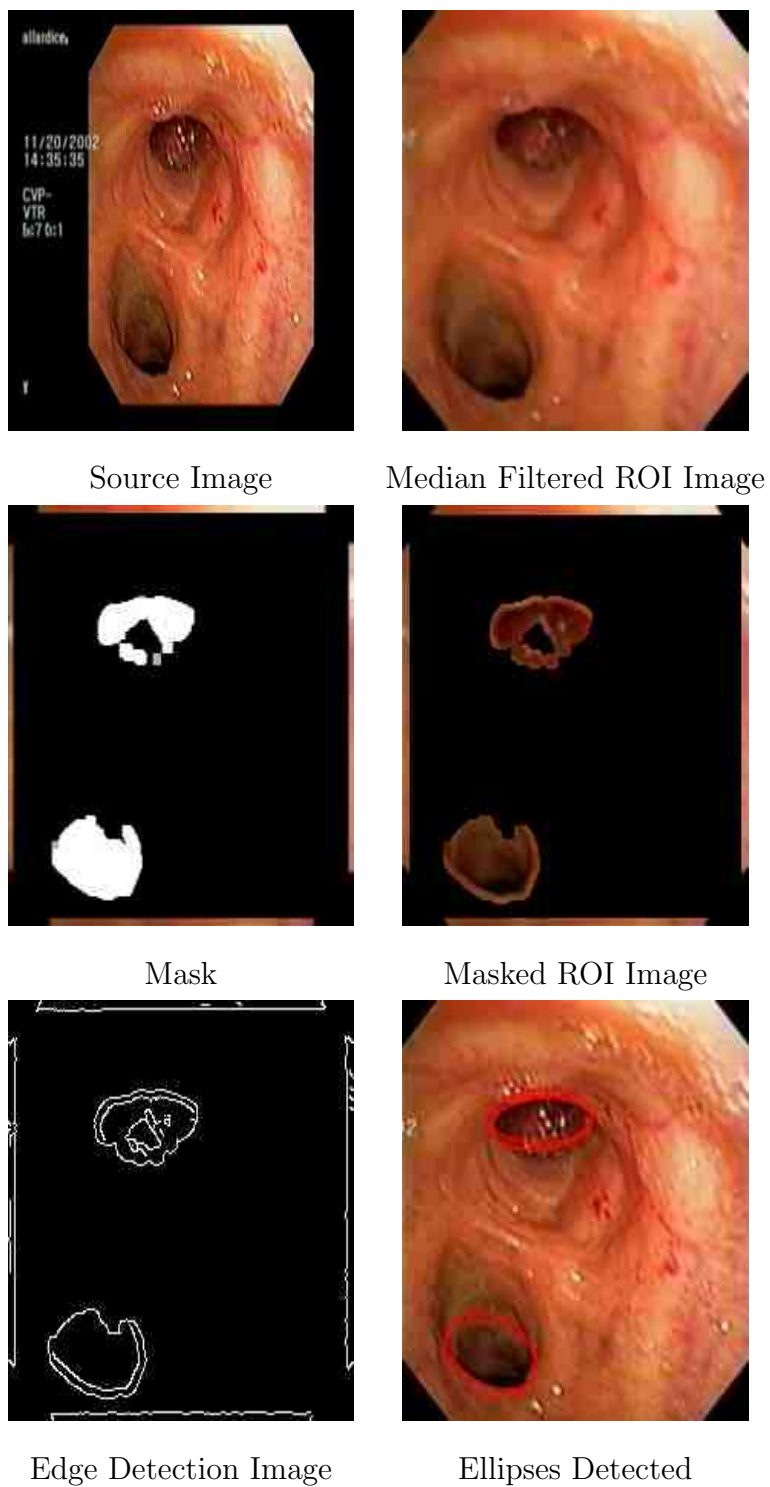


Fig. 2.11. Image #2: Example for two ellipses detected

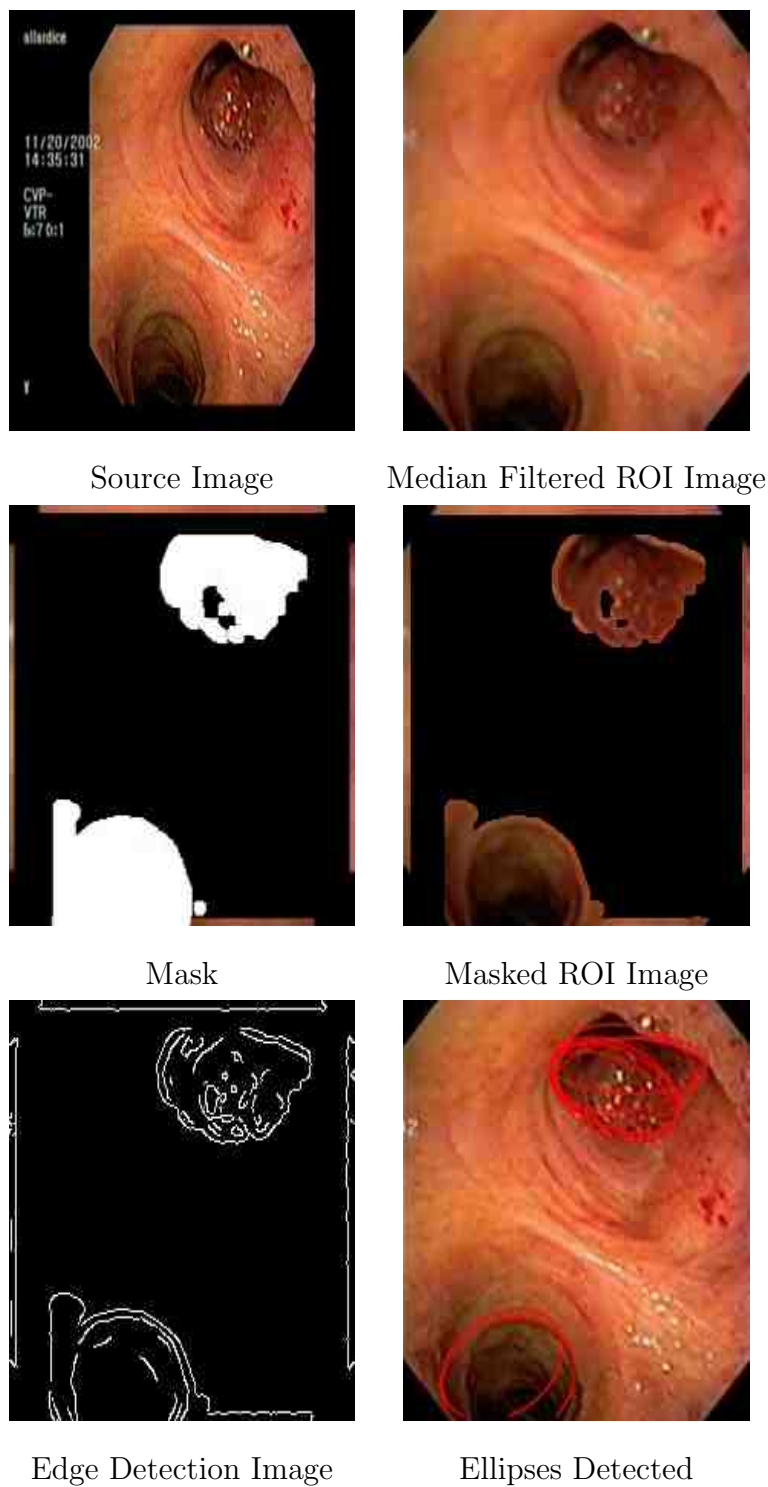


Fig. 2.12. Image #3: Example of ellipses detected from many edge pixels

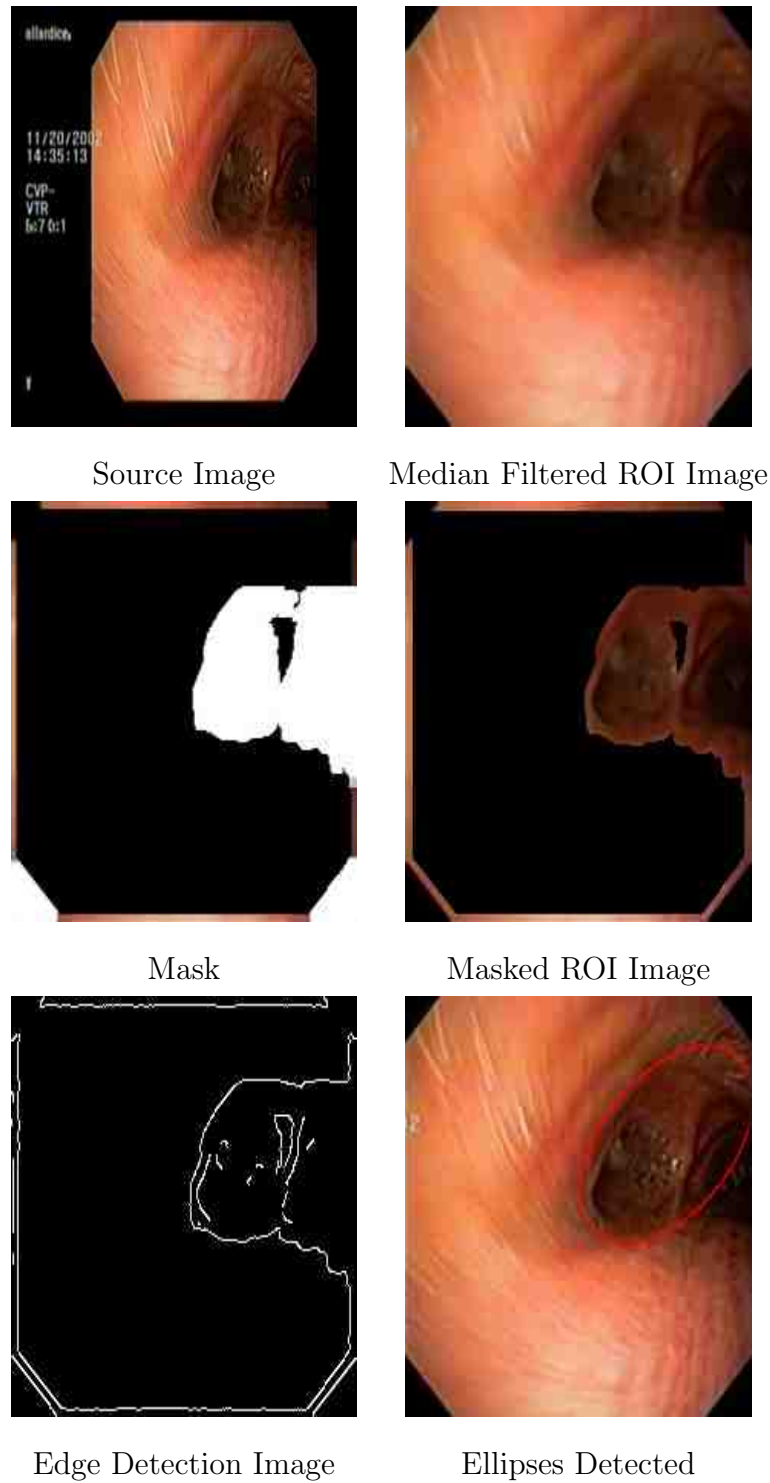


Fig. 2.13. Image #4: Example of one ellipse detected for two vanishing points present in source image

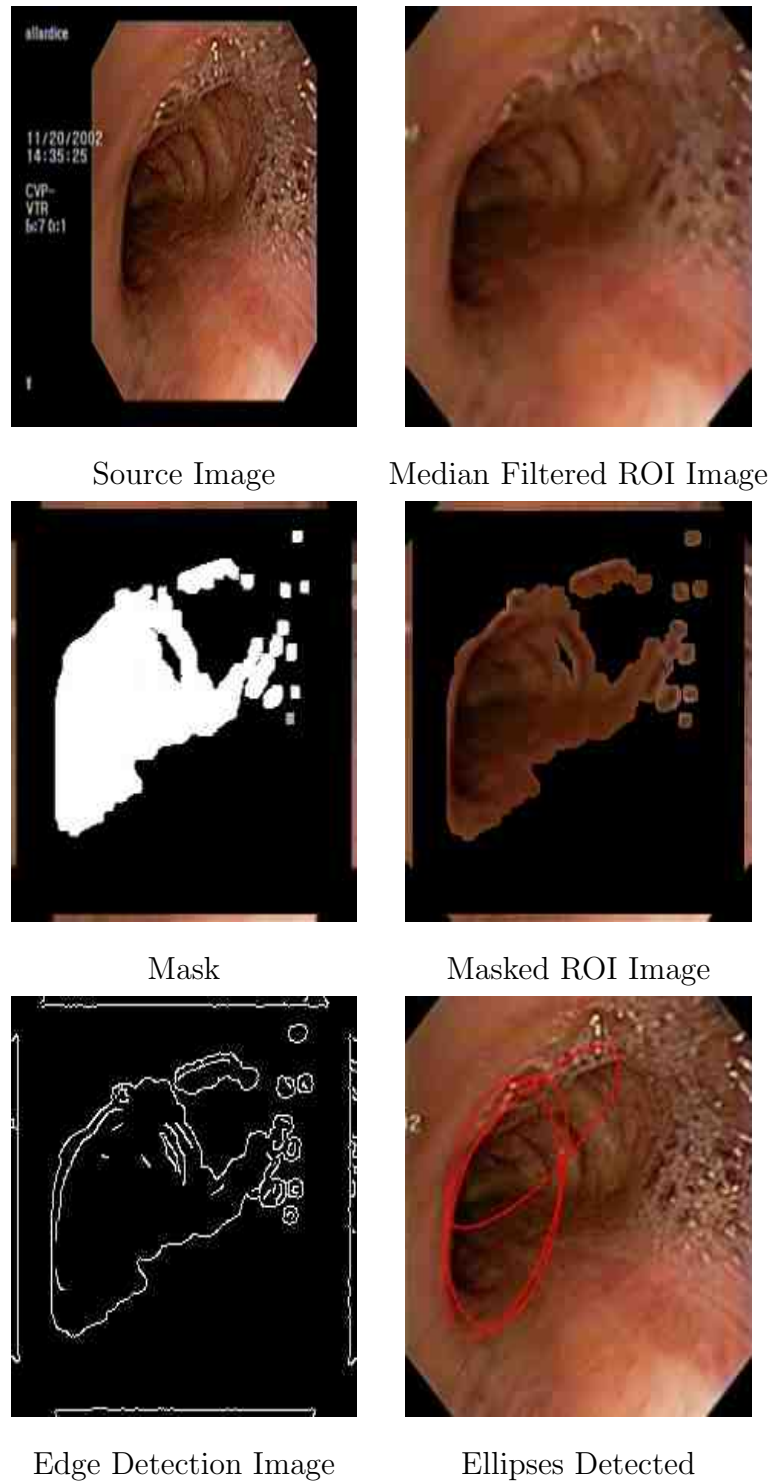


Fig. 2.14. Image #5: Example of ellipses detected with bubble clutter in source image

2.4 Depth-Map Generation

This is the last stage in the process of depth-map inference of 2D endoscopy image sequences. To create a depth gradient from one or more vanishing points has been adopted by a number of researchers, but it is restricted to indoor or outdoor hallway type images and has not been previously applied for 2D to 3D conversion of endoscopy videos. In [18], the images are first classified into 3: indoor, outdoor and outdoor with geometric elements such as buildings, bridges, etcetera, and based on this classification, vanishing lines and vanishing points are detected. Based on the slopes of these vanishing lines and the position of vanishing point, the depth-gradient planes are generated. The grey levels in these depth-gradient planes are based on assumptions that the vanishing point is the farthest point from the observer and that higher depth corresponds to lower grey values. In their paper, the grey levels in the horizontal planes are set to be constant along rows and the grey levels in vertical planes are set to be constant along columns. The two assumptions made in [18] are first implemented to estimate and generate a depth map for tunnel images consisting of a single vanishing point. This vanishing point is detected using centers of circles detected from gray scale images using Hough transform.

In this thesis, the depth gradient is grown such that the darkest grey level starts from the detected vanishing point and becomes lighter as it approaches the borders of the source image. For circles, this is illustrated in Figure 2.15. In this image, the circles in red denote the circles detected from our ellipse detection algorithm. The points in green are the centers of these circles and the single point in dark red is the point where the average of the centers of detected circles lies. This dark red point is considered as the vanishing point. To the right of the source image superimposed by the detected circles, their centers and the final vanishing point, is the depth gradient generated for the input tunnel image from the inferred vanishing point. The above method of gradient generation is used as the final step of our algorithm to develop a depth-map for endoscopy image sequences. After ellipses are detected from the edge



Fig. 2.15. Example of depth gradient generated (right) on tunnel image from circle detection (left)

segmented image of each endoscopy image, the detected ellipses are grouped into two sets since this particular set of endoscopy images have two vanishing points. The median value of x and y coordinates of the centers of ellipses detected is found and the distance between this median center and every other center is calculated. If this distance is less than a particular threshold, then the ellipses associated with those centers are considered as one set, and the ellipses whose centers lie at a distance greater than the threshold from the median value are considered as the second set. A mean of the centers of ellipses in each of the two groups is calculated, which gives the approximate location of the two vanishing points in the endoscopy image.

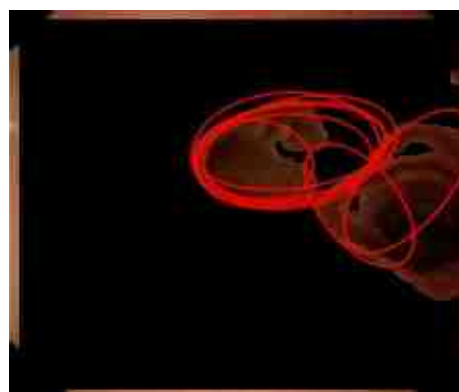
If only a single ellipse is detected by the ellipse detection algorithm, the center of that ellipse is taken as one of the vanishing points. The second vanishing point is obtained by finding the mean of points that fit a certain range of thresholds of red, blue and green components (50, 15 and 15 respectively) in the masked image. This is a just-in-case approach to find one of those vanishing points which is not detected by the algorithm in some of the endoscopy images due to reasons like insufficient edge pixels or unsuitability of the voting threshold. This approach to locate a possible vanishing point is used in 5% of the frames tested from the real bronchoscopy videos.

Figures 2.16, 2.17 and 2.17 show the two vanishing points detected for an endoscopy image from the database and the corresponding depth gradients generated by this algorithm. The blue ellipses in "vanishing point" images are the ellipses detected after grouping the outputs from ellipse detection stage into two groups. The green dots represent centers of ellipses and the vanishing points found when one of the ellipses is not inferred.

Once the depth gradients corresponding to the two vanishing points are generated, they are linearly added to obtain the final depth gradient. This final depth gradient is set back to its original dimensions before cropping out the region of interest in stage 1 and is stitched by the right side of the source image for 2D plus depth image based rendering [19].



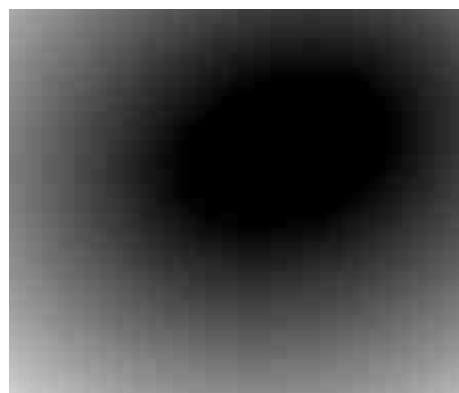
SourceImage



Ellipses Detected



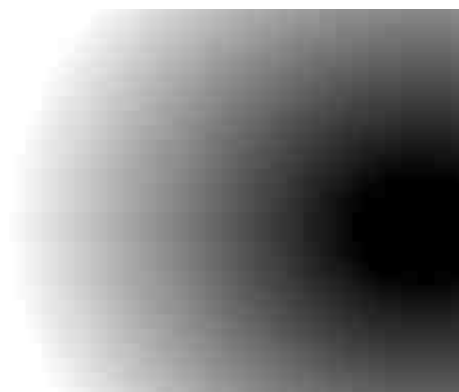
Vanishing Point 1 (VP1)
(First from left to right in image)



Corresponding Depth Gradient for VP1



Vanishing Point 2 (VP2)
(Second from left to right in image)



Corresponding Depth Gradient for VP2

Fig. 2.16. Depth-map generation example #1: Two ellipses detected



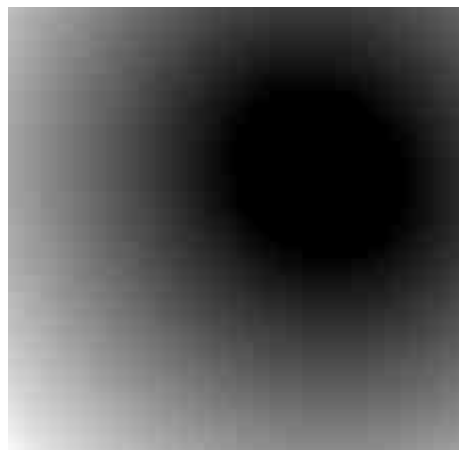
SourceImage



Ellipses Detected



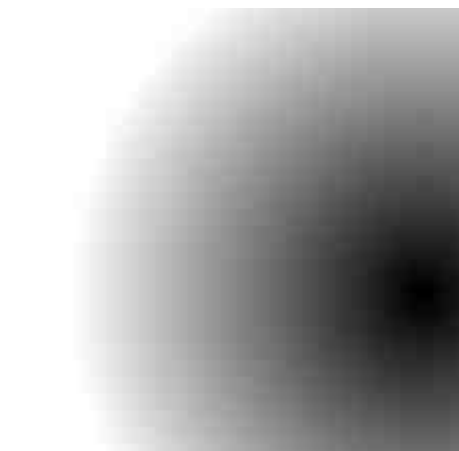
Vanishing Point 1



Corresponding Depth Gradient for VP1



Vanishing Point 2



Corresponding Depth Gradient for VP2

Fig. 2.17. Depth-map generation example #2: One ellipse detected, one inferred

3. RESULTS FROM REAL ENDOSCOPY IMAGES: COMPARING OUR RESULTS WITH TRUTH DATA

In this chapter, the resulting depth gradient obtained using our algorithm for endoscopy images is discussed. First, to test the credibility of the algorithm, it is tested on truth data. The truth images are obtained by developing gray scale gradients from known values of ellipse parameters. Then the algorithm is applied on these truth images to obtain the inferred depth-gradient. The mean square error (ε) between the truth images and their corresponding depth-maps inferred from the algorithm is calculated using Equation 3.1. Figure 3.1 shows the truth images in the first column and the depth-maps inferred in the second column. The ellipses detected in the truth images are superimposed on the inferred depth-map images. Table 3.1 compares the values of center coordinates, major axis, minor axis and orientation of ellipses of truth data and our algorithm. The mean square error between depth map generated from our algorithm and truth data is also listed in this table. The mean square error is in the range of 5 to 7 gray levels (highest being 255), which means that the depth map generated is very close to the actual gradient of the truth image.

$$\varepsilon = \frac{\sum_{i=1}^n (a_i - b_i)^2}{n} \quad (3.1)$$

where a_i and b_i represent pixel value of the i^{th} pixel in an array consisting of n elements in total.

Next, the algorithm is applied on the real endoscopy images from Dr. Noor. The endoscopy video of the first procedure (without instruments visible in the surgical site), when split into a stack of images at the rate of 29.97 frames per second gives a total of 846 images. Out of these, our algorithm is tested on 570 images to generate

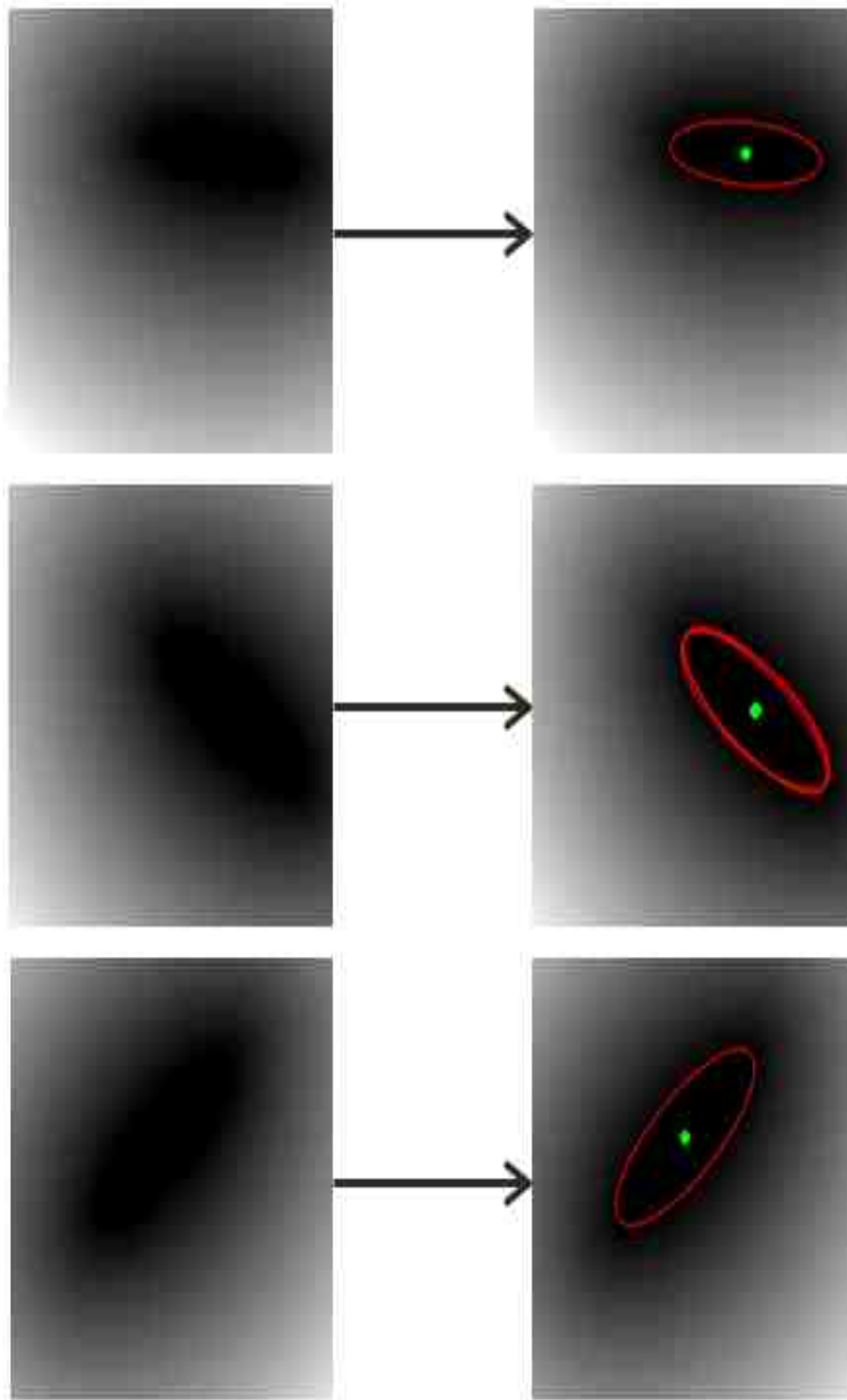


Fig. 3.1. Truth image (left) and inferred depth with detected ellipses superimposed (right)

Table 3.1
Comparison of ellipse parameters of truth images and inferred depths

Image #		Parameters					Mean Square Error (ε)
		x	y	a	b	α	
1	Truth	149	69	55	16	-6	5
	Inferred	149	69	53	15	-4	
2	Truth	157	108	62	23	-36	7
	Inferred	156	108	60	22	-33	
3	Truth	107	87	63	22	41	6
	Inferred	107	86	61	21	40	

their depth maps. Figures 3.2 to 3.4 show three of the input images from the first procedure as examples.

The video of the second procedure given by Dr. Noor is divided into sequence of images at the rate of 29.97 frames per second, resulting in a total of 1489 images. Out of these, our algorithm is tested on 450 images and their depth map is generated. The region of interest for these images has dimensions 215×304 . The thresholds for darkest area detection are not changed. Also, the thresholds for Canny edge detection are the same as described in the above sections. The advantage of our algorithm is that the presence of surgical instruments in the in-vivo surgical site does not affect the detection of vanishing point(s) in the image; this is because it is observed that the surgical tools are usually brighter than the darkest regions of the image. Due to this, the regions of the image where the surgical tools are present get masked after the darkest area detection step in stage 2. Figures 3.5 to 3.7 show three examples of images from the second procedure and the results of application of our algorithm on these input source images. Figures 3.2 to 3.7 contain the input source image, the image containing the detected vanishing points, the corresponding depth gradients for each vanishing point detected and the final depth-map corresponding to the input source image.

With respect to depth map generated for endoscopy images from the first procedure, Dr. Aliya Noor stated that subjective viewing of the 3D effect in the bronchoscopy video using the depth map inferred by our algorithm, appears natural and could be useful in a clinical setting with more testing.

More results of our algorithm implemented on the endoscopy videos from the first and second bronchoscopy videos given by Dr. Noor are included in appendix A. Section 1 of appendix A contains some of the results that have two ellipses detected for the two vanishing points while section 2 contains results of images with two vanishing points but one from detected ellipse and the other from inferred vanishing point and input images with single vanishing point. The images in sections 1 and 2 of appendix A are representative of a number of frames of the original endoscopy video.

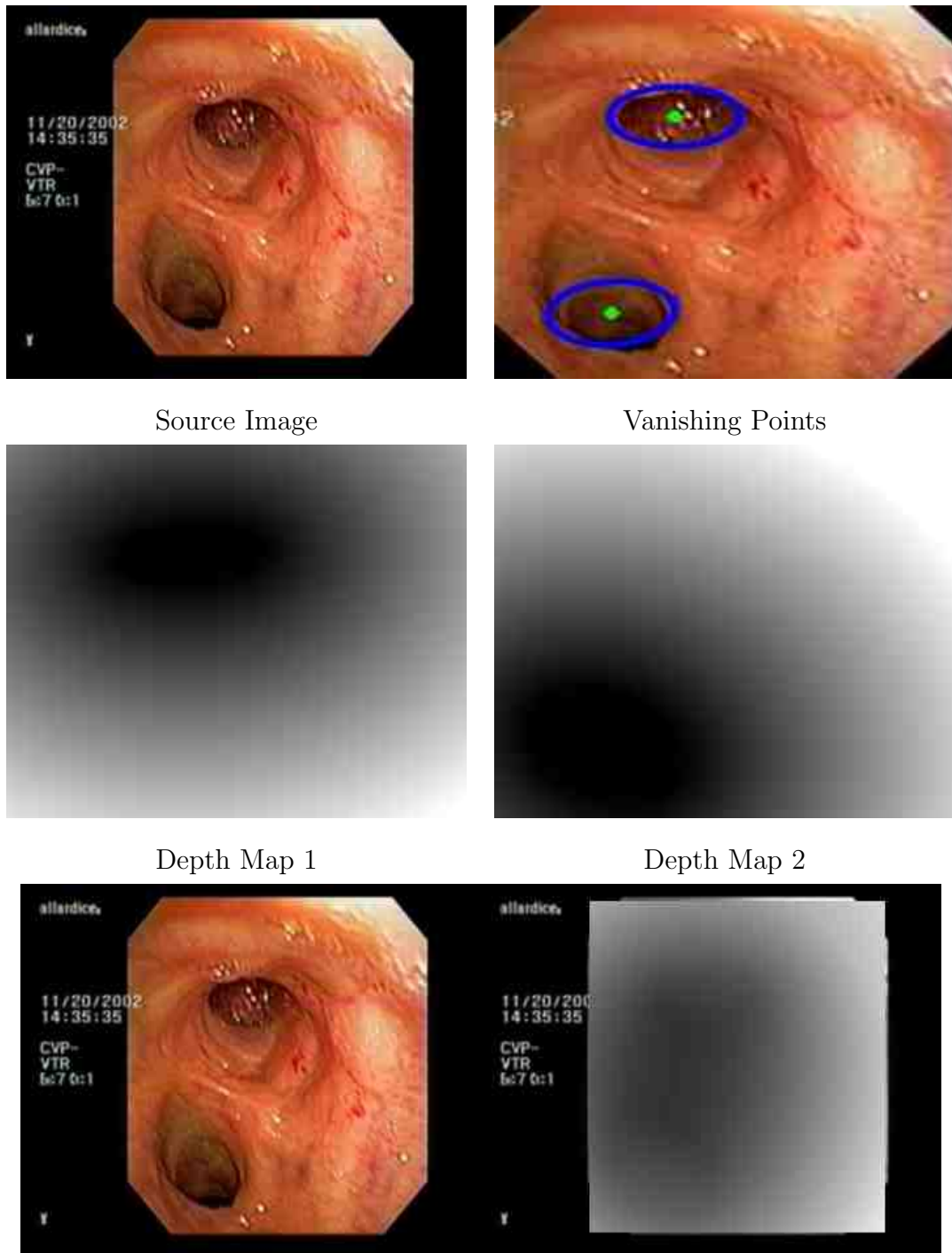


Fig. 3.2. Depth gradient obtained for image #1: without surgical instrument

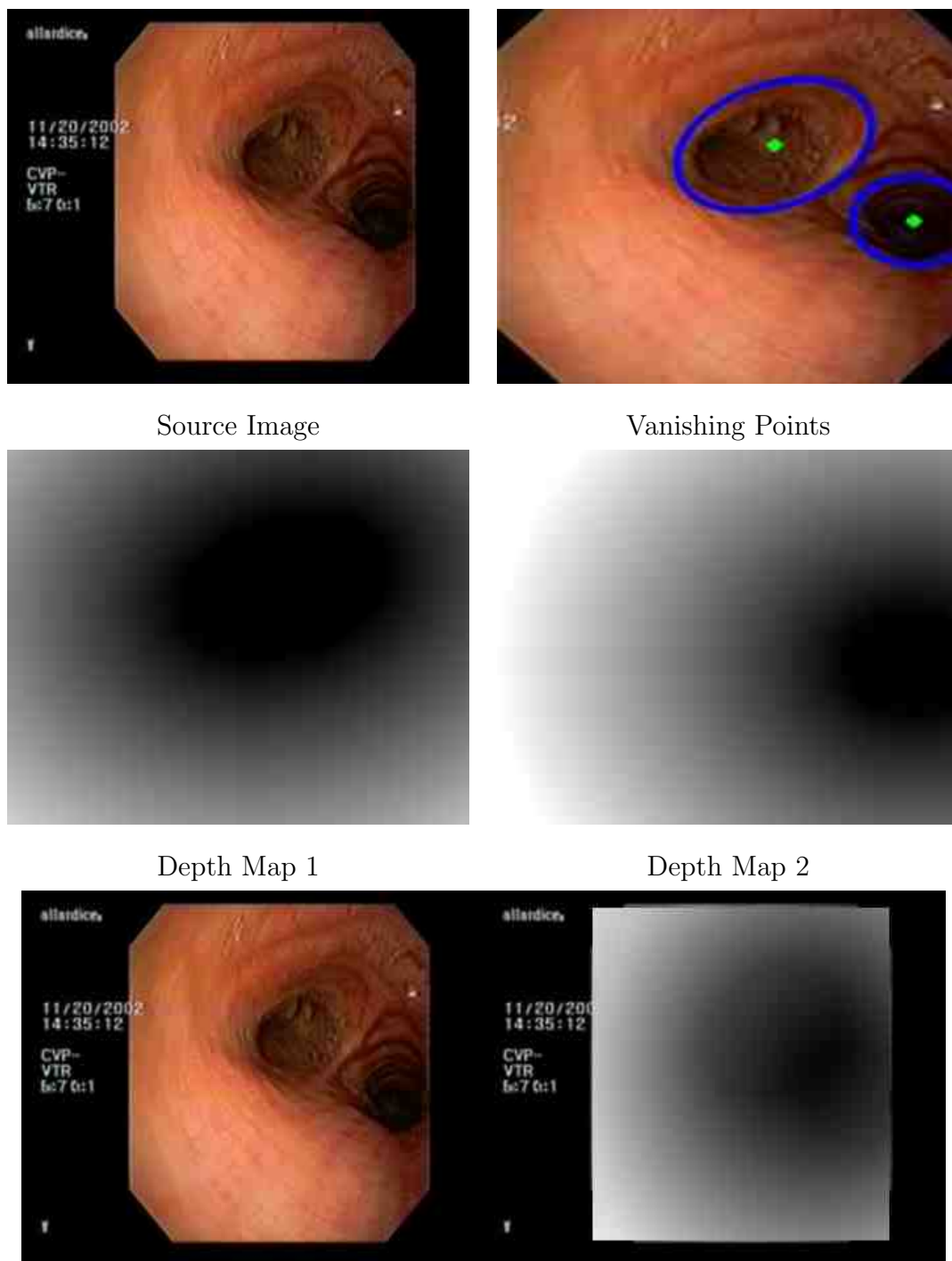


Fig. 3.3. Depth gradient obtained for image #2: without surgical instrument

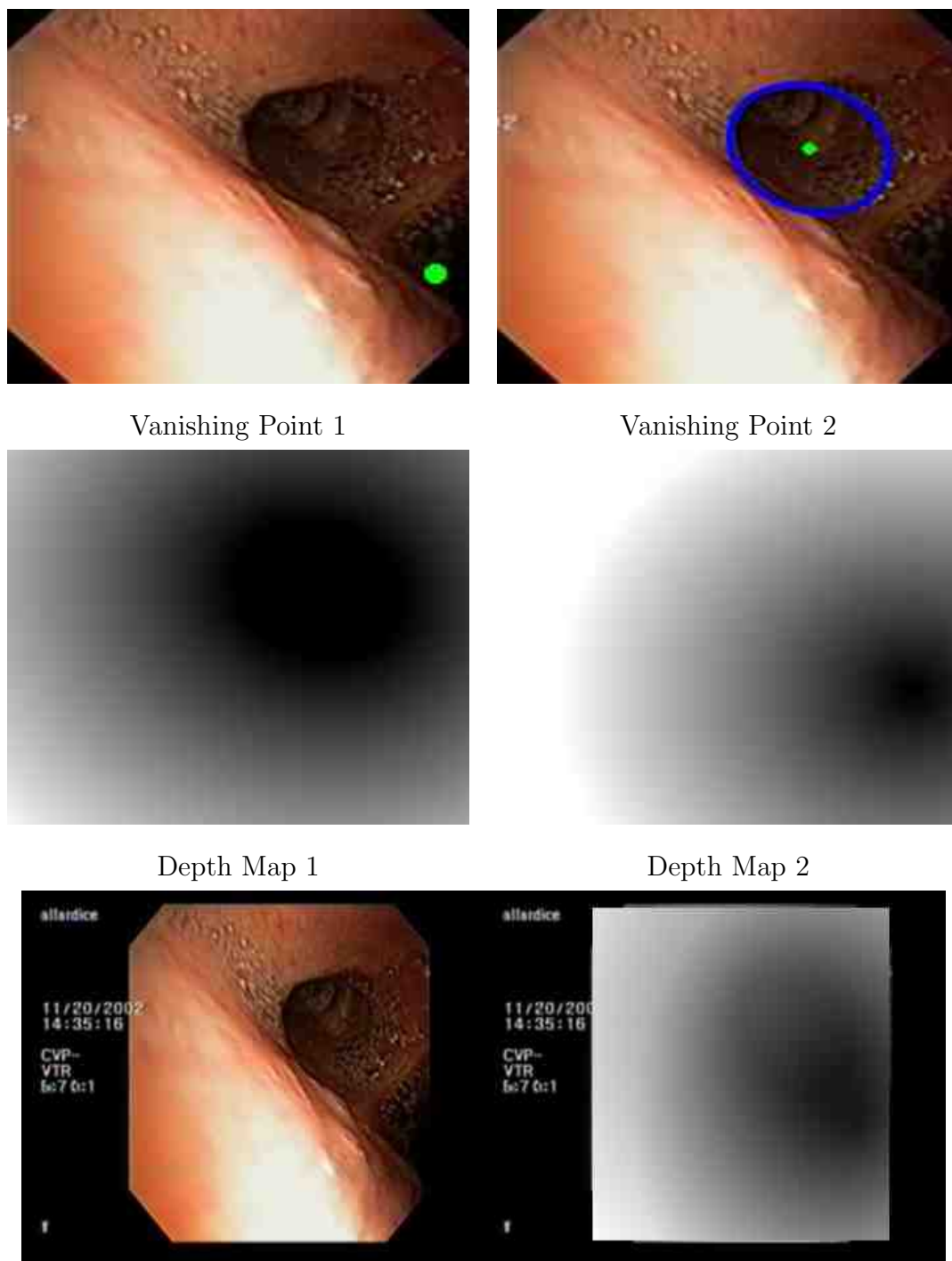


Fig. 3.4. Depth gradient obtained for image #3: without surgical instrument

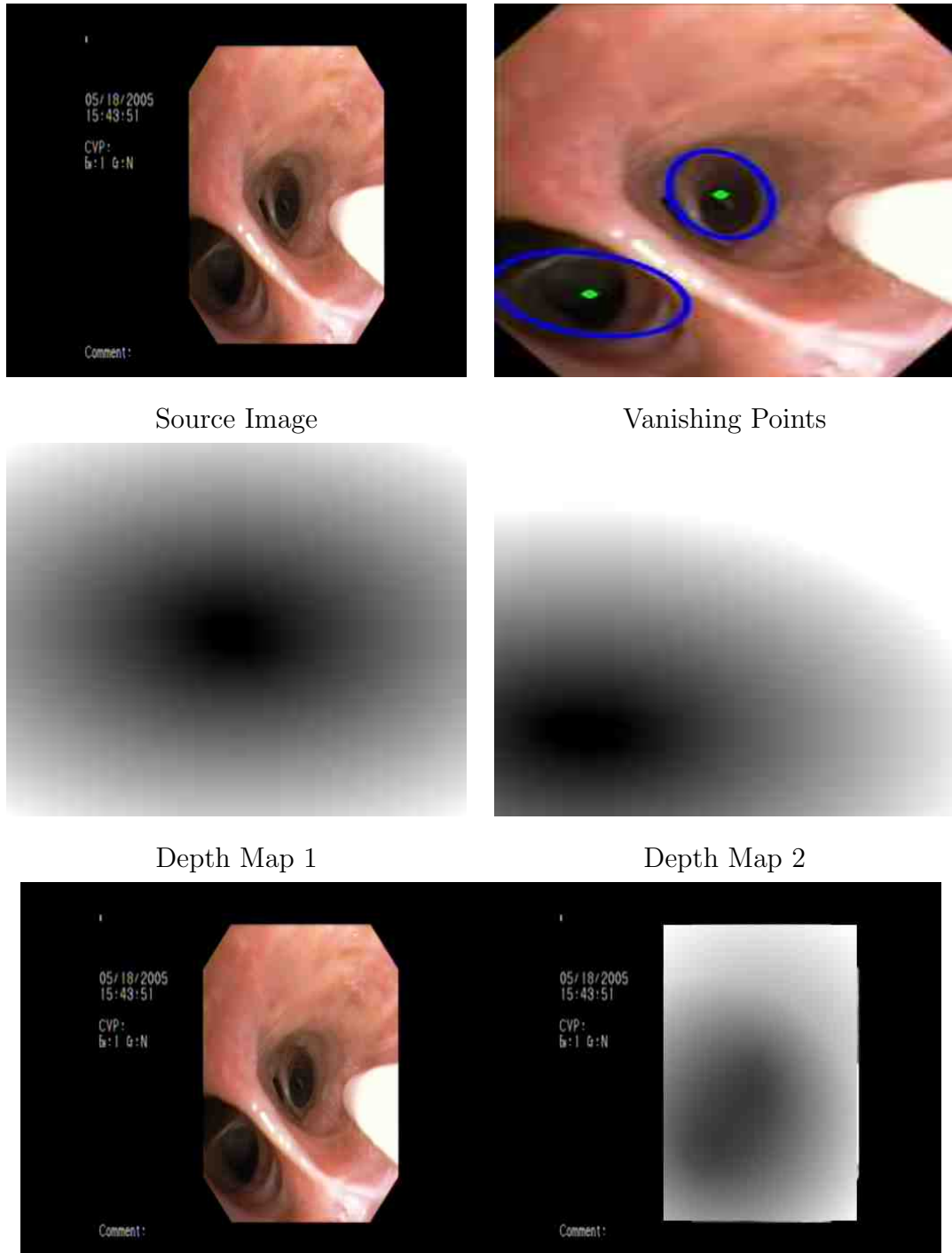
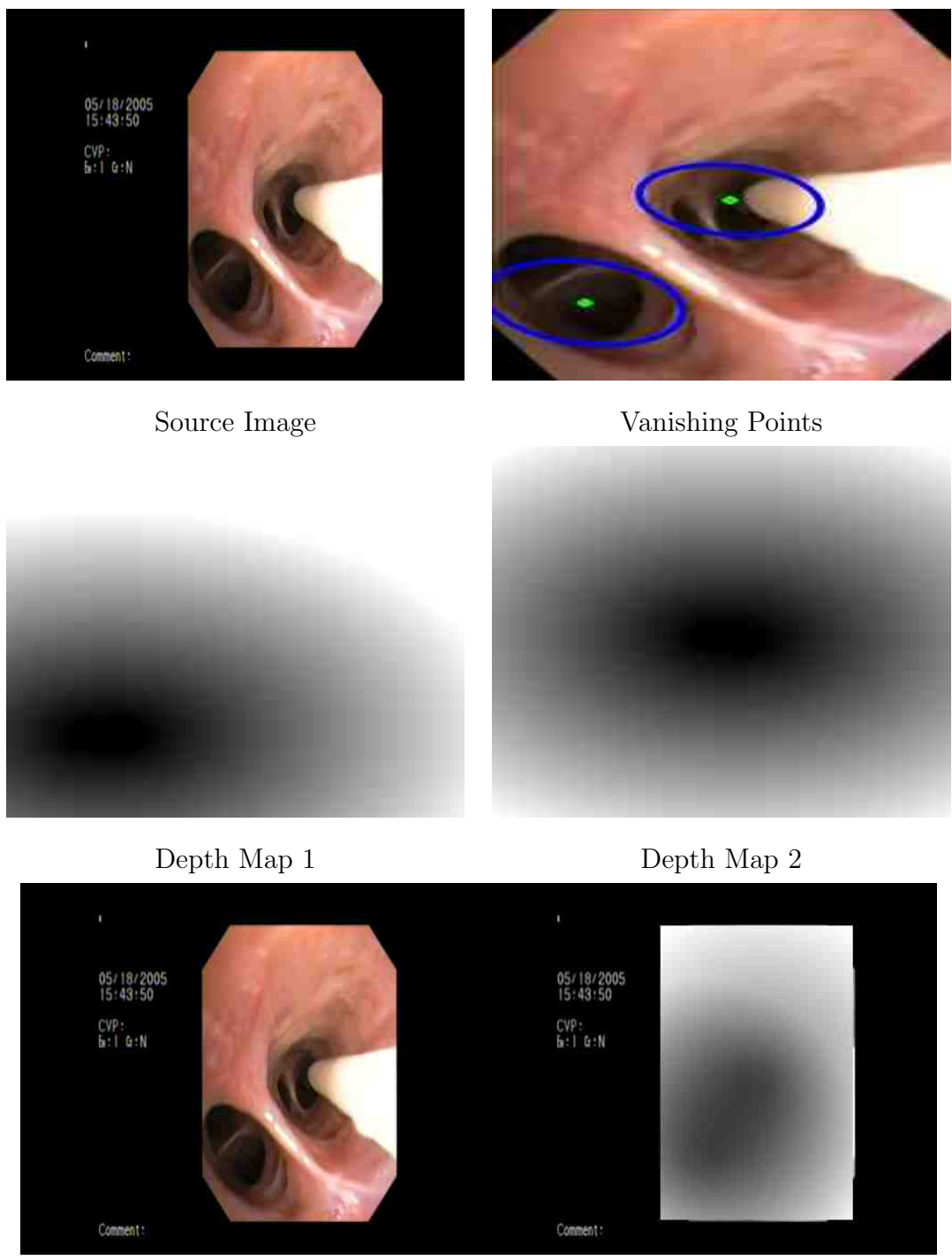


Fig. 3.5. Depth gradient obtained for image #1: with surgical instrument



Source Image

Vanishing Points

Depth Map 1

Depth Map 2

Fig. 3.6. Depth gradient obtained for image #2: with surgical instrument

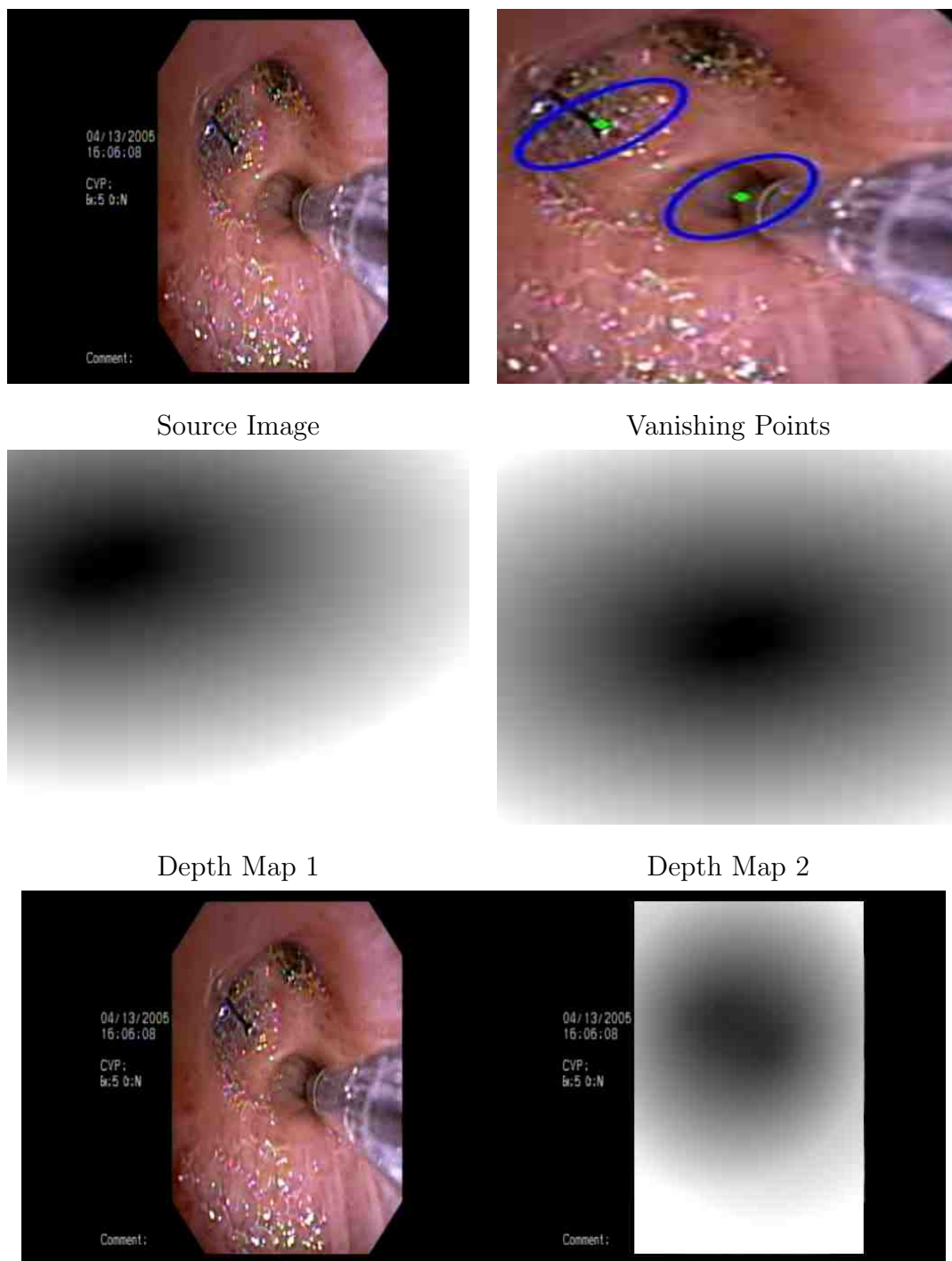


Fig. 3.7. Depth gradient obtained for image #3: with surgical instrument

For the 570 test images from procedure one, our algorithm detected two vanishing points from two detected ellipse parameters in 225 of these images and from one detected ellipse and one inferred point in 331 of the images. For the 450 test frames of the second procedure, two vanishing points were detected from two detected ellipse parameters in 97 frames and single vanishing point was detected in 322 frames.

Table 3.2 gives the execution times of each stage of our algorithm on six representative images from the endoscopy image sequence for varying edge pixels. These runtimes are recorded when the algorithm is executed on a computer with 2.27GHz clock frequency, 4GB RAM and Intel Core i3 CPU. It can be observed that the time of execution varies between 21 seconds to 20 minutes. The execution time of the third stage, i. e., ellipse detection stage is the reason for increase in running time of the algorithm, as the ellipse detection code searches through each edge pixel three times to detect the best ellipse. The execution time of the ellipse detection stage is proportional to the number of edge pixels. Figure 3.8 shows a graph of variation of execution time of our algorithm with number of edge pixels.

Table 3.2
Time elapsed for execution of each stage of our algorithm (seconds)

Image #	# of edge pixels	Time Elapsed (in seconds)				
		Stage 1	Stage 2	Stage 3	Stage 4	Total
1	1289	0.014	0.387	20.043	0.616	21.060
2	1366	0.017	0.384	71.018	0.612	72.031
3	1552	0.016	0.374	89.139	0.592	90.121
4	2047	0.008	0.387	186.246	0.529	187.17
5	2305	0.000	0.375	599.118	0.686	600.179
6	2627	0.000	0.546	1247.314	0.890	1248.750

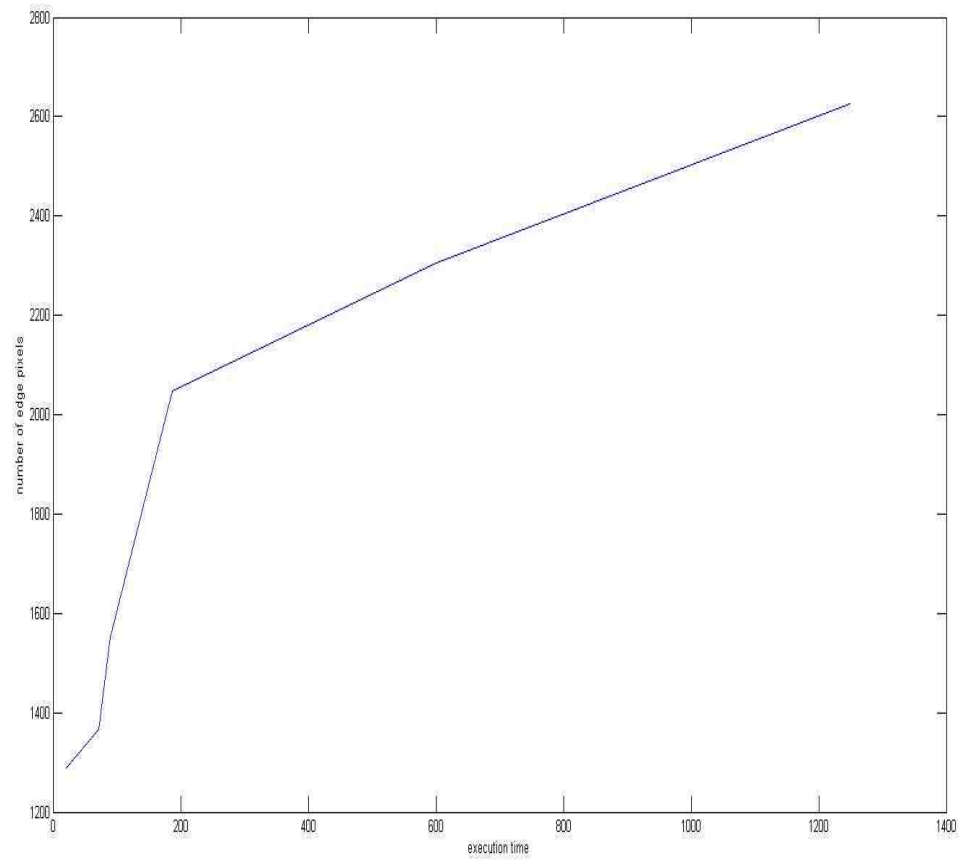


Fig. 3.8. Graph representing execution time of our algorithm vs number of edge pixels

4. CONCLUSION AND FUTURE WORK

In this thesis, a novel algorithm was proposed and developed to convert 2D endoscopy video to 3D using depth inference. Interpretation of depth was done by finding the vanishing points in 2D endoscopy image sequence obtained from detection of the centers and orientations of elliptical ridges present in the edge detection images. The depth-map was generated by growing a depth gradient from these vanishing points. This is a new approach that has never been employed by researchers previously for depth-map inference of monocular 2D endoscopy videos to convert them to 3D.

In Chapter 1, the motivation to start this research was explained by stating the advantages of 3D endoscopy and the previous work related to conversion of 2D endoscopy videos to 3D. In Chapter 2, the algorithm developed by us was described stage-wise with examples. In Chapter 3, the locations of vanishing points detected from our algorithm were compared with truth data. Results of application of our algorithm on real endoscopy images with and without the presence of surgical instruments in the site of surgery were also discussed and illustrated in this chapter. Also, in Chapter 3, the execution time of each stage of our algorithm for different number of input edge pixels were tabulated and their causes and effects were discussed.

The credibility of our algorithm was first tested on synthetic truth images. The truth images were obtained by growing concentric ellipses starting from the edges of an ellipse with known parameter values. Our algorithm was then implemented on these truth images and the corresponding vanishing point was found and depth gradient was generated. The mean squared error between each pixel value of the truth image and obtained depth gradient image was calculated and found to have values of 5, 6 and 7 gray levels, the highest value of gray level being 255. The % accuracy of ellipse detection was found to be in the range of 98.98% to 99.99%.

For the first procedure, where no surgical instruments were present in the endoscopy video, 570 frames out of a total of 846 frames were tested with our algorithm. In the second procedure consisting of surgical instruments, 450 of a total of 1489 frames were tested. The number of frames where desired ellipses and vanishing points were detected by our algorithm are tabulated in Table 4.1. The results from Table 4.1 show that the desired single and double vanishing points are located from our algorithm for the first and second procedures in 92% to 97.89% of the total number of frames tested. These values are calculated using Equation 4.1.

Table 4.1
Comparing accuracy of our algorithm with real values

Procedure # (# of frames tested)	# of frames with 1 vanishing point			# of frames with 2 vanishing points		
	Real	Detected	% Accuracy	Real	Detected	% Accuracy
1(570)	338	331	97.89	232	225	96.89
2(450)	350	322	92	100	97	97.4

$$\%Accuracy = 100 - \left| \frac{real - detected}{real} \right| \quad (4.1)$$

From Table 4.1 it can be seen that out of a total of 1020 frames tested (combining procedures 1 and 2), single vanishing point was detected as desired in 653 out of 688 frames (94.9% of the frames) and two vanishing points were detected in 322 out of 332 frames (96.98% of the frames).

The limitation of our approach is the execution time of the algorithm. It can be observed from Table 3.2 that as the number of edge pixels increases, the execution time of the algorithm increases. It can also be observed from this table that stage 3, which is the ellipse detection stage, takes the maximum time for execution, which is the reason for high execution time of the algorithm. The reason for high execution time of the ellipse detection stage is because the algorithm performs an exhaustiv

search of each edge pixel, three times. Hence, as future work, the execution time of the algorithm can be reduced by implementing an ellipse detection algorithm that has lesser time complexity. Also, another method to reduce the execution of the overall algorithm is by implementing it with the aid of OpenMP parallel processing. Further improvement of the part of the algorithm to group the ellipses into two groups to detect the two vanishing points present in bronchoscopy videos can be done using k-means clustering algorithm [20]. In addition, the work for future would be to test the algorithm on more endoscopy image sequences from various other sources.

A real time version of this algorithm can be developed by designing a hardware device that takes input video from the 2D endoscopy camera and executes this algorithm to give 2D plus depth video as output. For this, the speed of execution must be at the rate of 0.02 frames per second. This can be achieved by parallel processing of the microprocessors or a suitable hardware design.

Other developments that can be made to this algorithm are as follows:

- Allowing the user to change the thresholds for the red, green and blue components in the image for darkest area detection and the upper and lower thresholds of Canny edge detection.
- Using this algorithm together with other algorithms such as structure-from-motion, depth-from-defocus, etc., to infer a more accurate depth-map for endoscopy images.
- Using knowledge of locations of vanishing points in previous frames of the video to interpret the location of vanishing points in the current frame.
- Improve the method of combining depth gradients of the two vanishing points detected, to make a more accurate depth gradient.

Our algorithm is a novel approach to infer the depth-map of endoscopy images, irrespective of the type of camera used to capture the video. In addition, this algorithm is automatic for each set of input source image sequence. The percentage accuracy of identification of vanishing points by our algorithm from Table 4.1 shows that this algorithm can be implemented successfully in endoscopy videos without instruments and in bronchoscopy videos with surgical instruments by making slight changes in it.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] R. P. Manes, S. Barnett, and P. S. Batra, "Utility of novel 3-dimensional stereoscopic vision system for endoscopic sinonasal and skull-base surgery," *International Forum of Allergy and Rhinology*, vol. 1, no. 3, pp. 191–197, 2011.
- [2] N. Taffinder, S. G. Smith, J. Huber, R. C. Russell, and A. Darzi, "The effect of a second-generation 3d endoscope on the laparoscopic precision of novices and experienced surgeons," *Surgical Endoscopy*, vol. 13, no. 11, pp. 1087–1092, 1999.
- [3] J. C. Byrn, S. Schluender, C. M. Divino, J. Conrad, B. Gurland, E. Shlasko, and A. Szold, "Three-dimensional imaging improves surgical performance for both novice and experienced operators using the da vinci robot system.," *American journal of surgery*, vol. 193, no. 4, pp. 519–522, 2007.
- [4] K. Peitgen, M. V. Walz, M. V. Walz, G. Holtmann, and F. W. Eigler, "A prospective randomized experimental evaluation of three-dimensional imaging in laparoscopy," *Gastrointestinal Endoscopy*, vol. 44, no. 3, pp. 262 – 267, 1996.
- [5] R. Satava, "3-d vision technology applied to advanced minimally invasive surgery systems," *Surgical Endoscopy*, vol. 7, pp. 429–431, 1993.
- [6] C. R. Doarn, "Conference support - surgery in extreme environments - center for surgical innovation," p. 32, January 2007.
- [7] Q. Wei, "Converting 2D to 3D: A Survey." Research Assignment for Master Program Media and Knowledge Engineering of Delft University of Technology, December 2005.
- [8] K. Keller and A. State, "A single-imager stereoscopic endoscope," vol. 7964, p. 79641Z, SPIE, 2011.
- [9] M. Chan, W. Lin, C. Zhou, and J. Y. Qu, "Miniaturized three-dimensional endoscopic imaging system based on active stereovision," *Appl. Opt.*, vol. 42, pp. 1888–1898, April 2003.
- [10] C.-G. Song and J. U. Kang, "Design of the computerized 3d endoscopic imaging system for delicate endoscopic surgery," *J. Med. Syst.*, vol. 35, pp. 135–141, February 2011.
- [11] B. Linnane, G. M. Hafen, and S. C. Ranganathan, "Diameter of paediatric sized flexible bronchoscopes: When size matters," *Pediatric Pulmonology*, vol. 41, no. 8, pp. 787–789, 2006.
- [12] F. Mourgues, F. Devemay, and E. Coste-Maniere, "3d reconstruction of the operating field for image overlay in 3d-endoscopic surgery," in *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pp. 191 –192, 2001.

- [13] J. Zhou, Q. Zhang, B. Li, and A. Das, "Synthesis of stereoscopic views from monocular endoscopic videos," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pp. 55 – 62, June 2010.
- [14] T. Thormählen, H. Broszio, and P. Meier, "Three-dimensional endoscopy," *Falk Symposium No. 124, Medical Imaging in Gastroenterology and Hepatology, Hannover, Germany, September 2001, Kluwer Academic Publishers, 2002, ISBN 0-7923-8774-0*, vol. 0, no. 0, 2002.
- [15] P. V. C. Hough, "A Method and Means for Recognizing Complex Patterns." US Patent 3,069,654, 1962.
- [16] D. H. Ballard, "Generalized hough transform to detect arbitrary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 111–122, 1981.
- [17] Y. X. Qiang and Q. Ji, "A new efficient ellipse detection method," in *International Conference on Pattern Recognition 2002*, p. 20957, 2002.
- [18] S. Battiato A, S. Curti B, M. La Cascia C, M. Tortora C, and M. Scordato C, "Depth-map generation by image classification."
- [19] "3d interface specifications, white paper," *Philips Electronics 3D Solutions*, February 2008.
- [20] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

APPENDICES

A. ALGORITHM IMPLEMENTED ON MORE BRONCHOSCOPY IMAGES

The endoscopy video used to test this algorithm is available on the following website:

<http://www.engr.iupui.edu/lauchris/Assets/Bronchoscopy3D.avi>

A.1 Two vanishing points detected from two ellipses

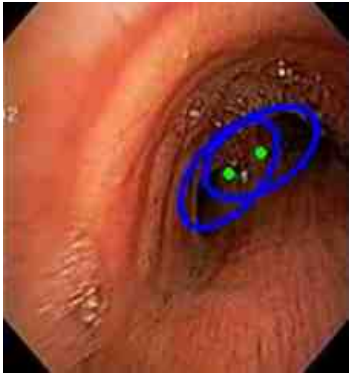
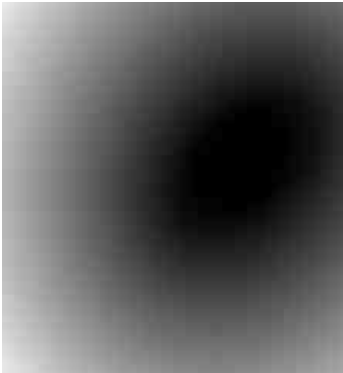

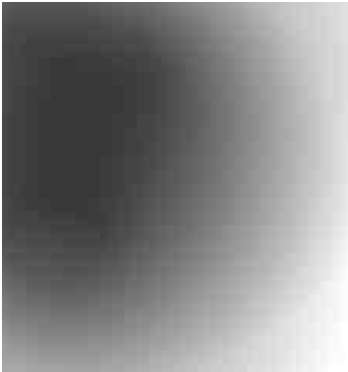
# of frames represented	Detected Ellipses and Vanishing Points	Depth-Map Generated
61/570		
51/570		

Fig. A.1. Result of algorithm implemented on dataset with two detected vanishing points from ellipses

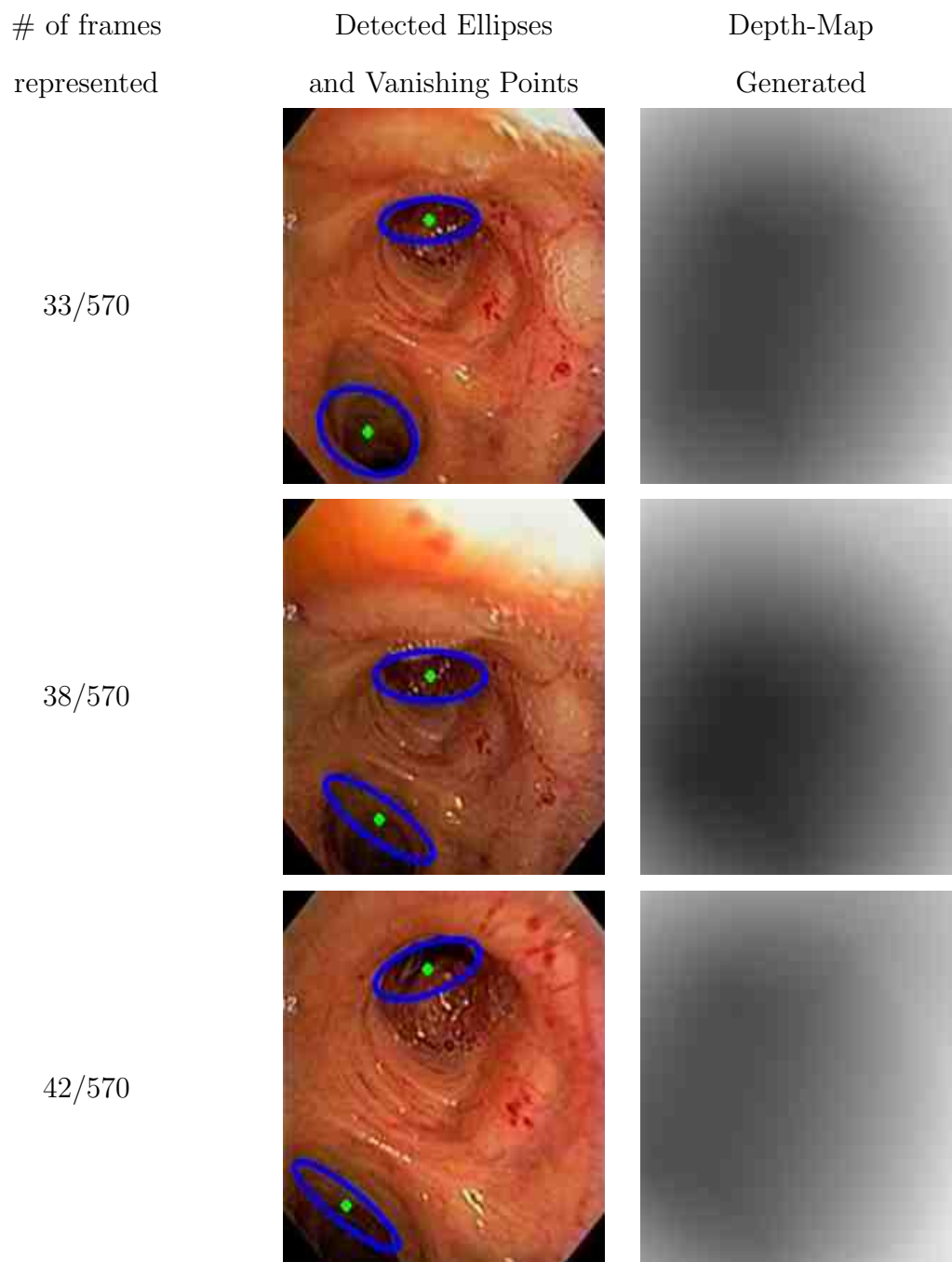


Fig. A.2. Result of algorithm implemented on dataset with two detected vanishing points from ellipses (continued from Figure A.1)

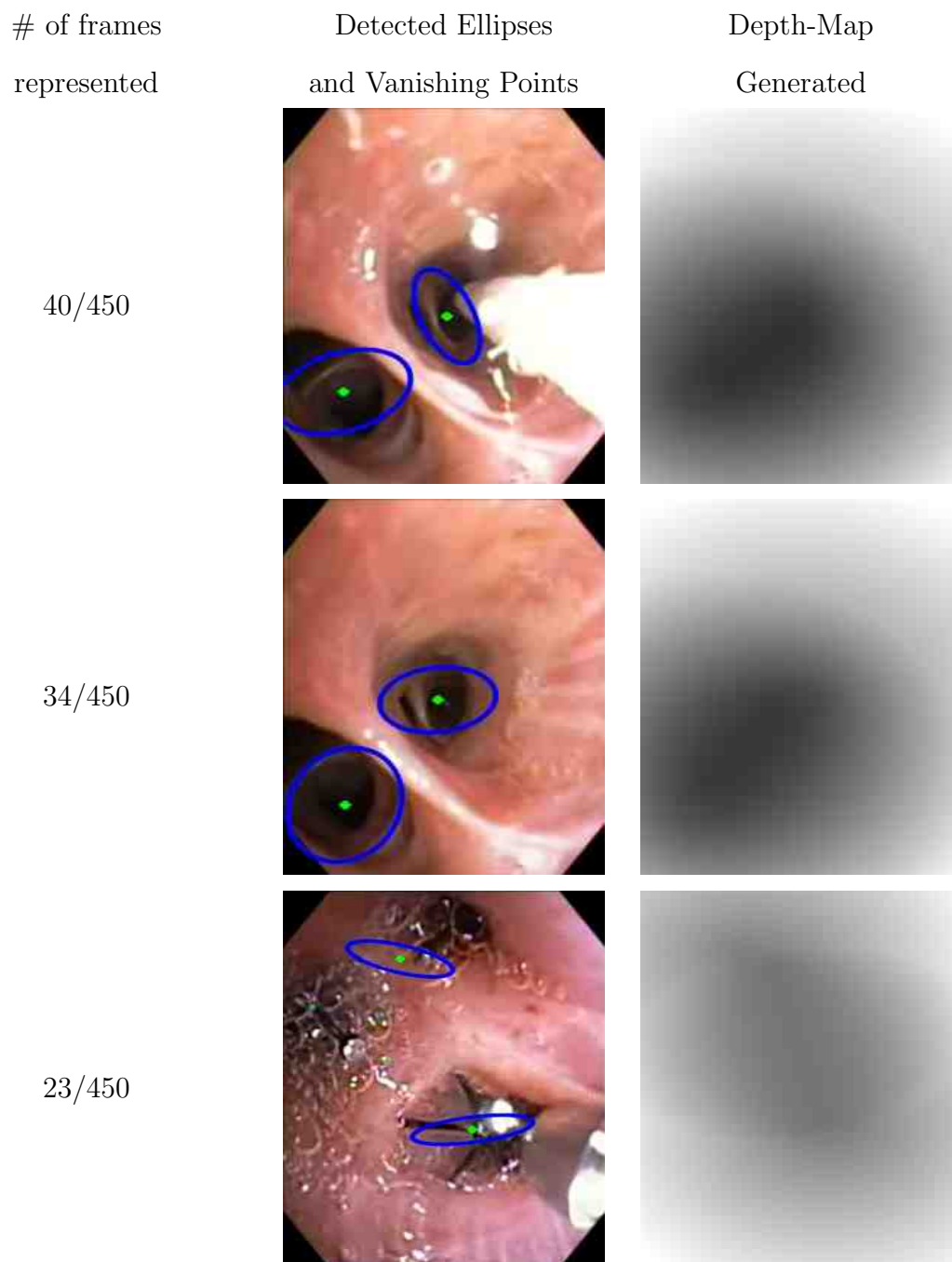


Fig. A.3. Result of algorithm implemented on dataset with two detected vanishing points from ellipses (continued from Figure A.2)

A.2 One vanishing point detected from ellipse and one inferred vanishing point

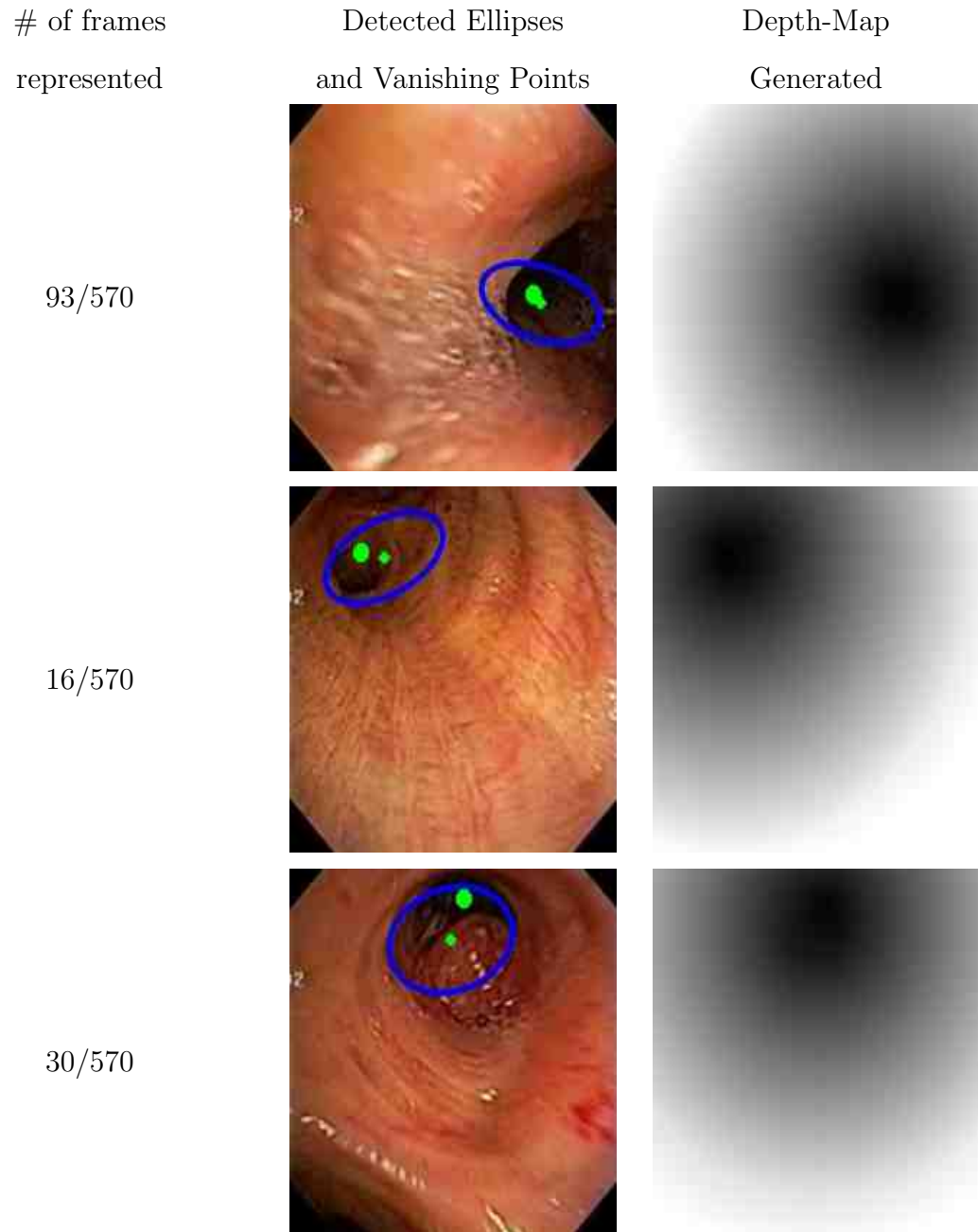


Fig. A.4. Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred

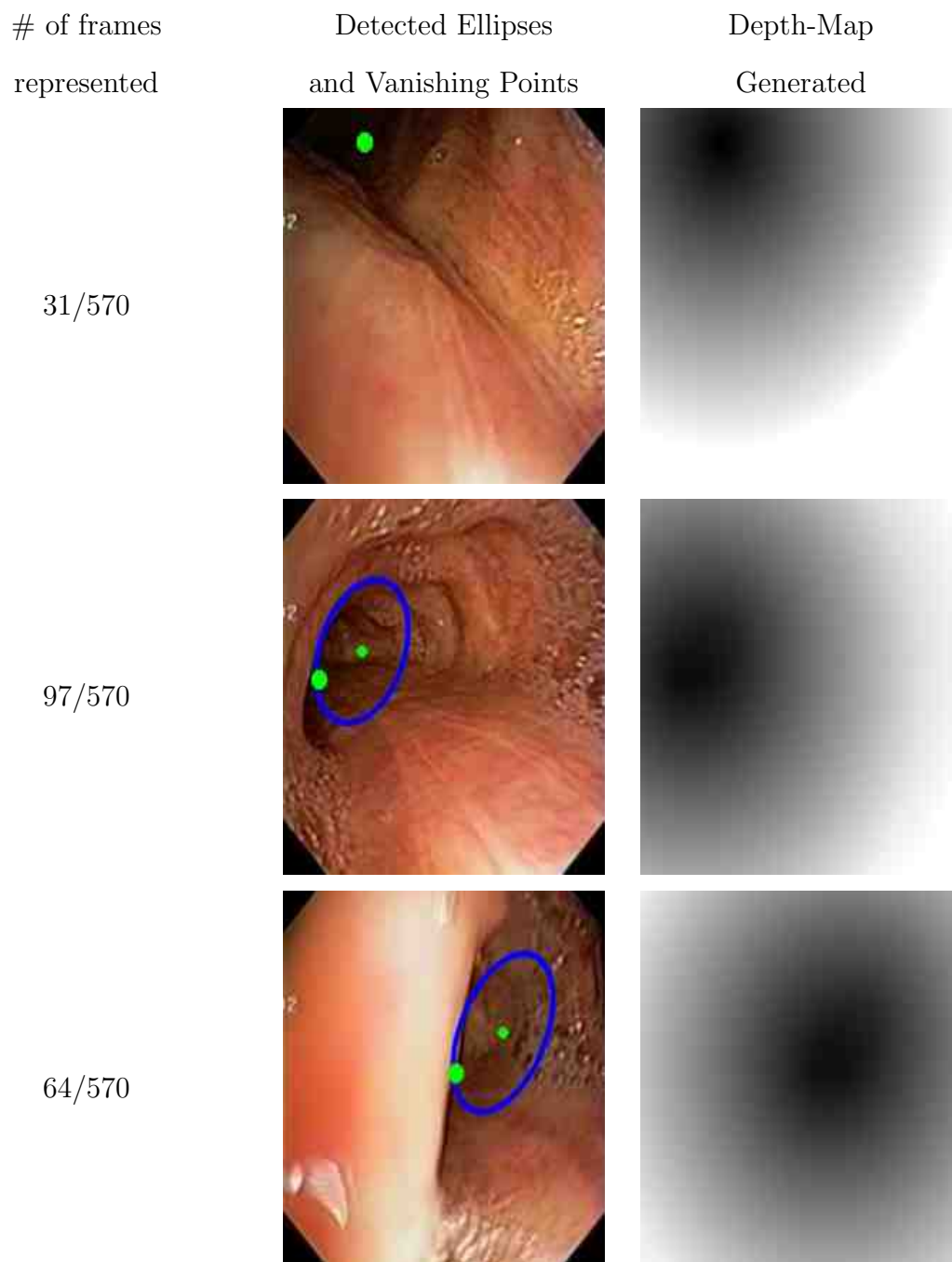


Fig. A.5. Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred (continued from Figure A.4)

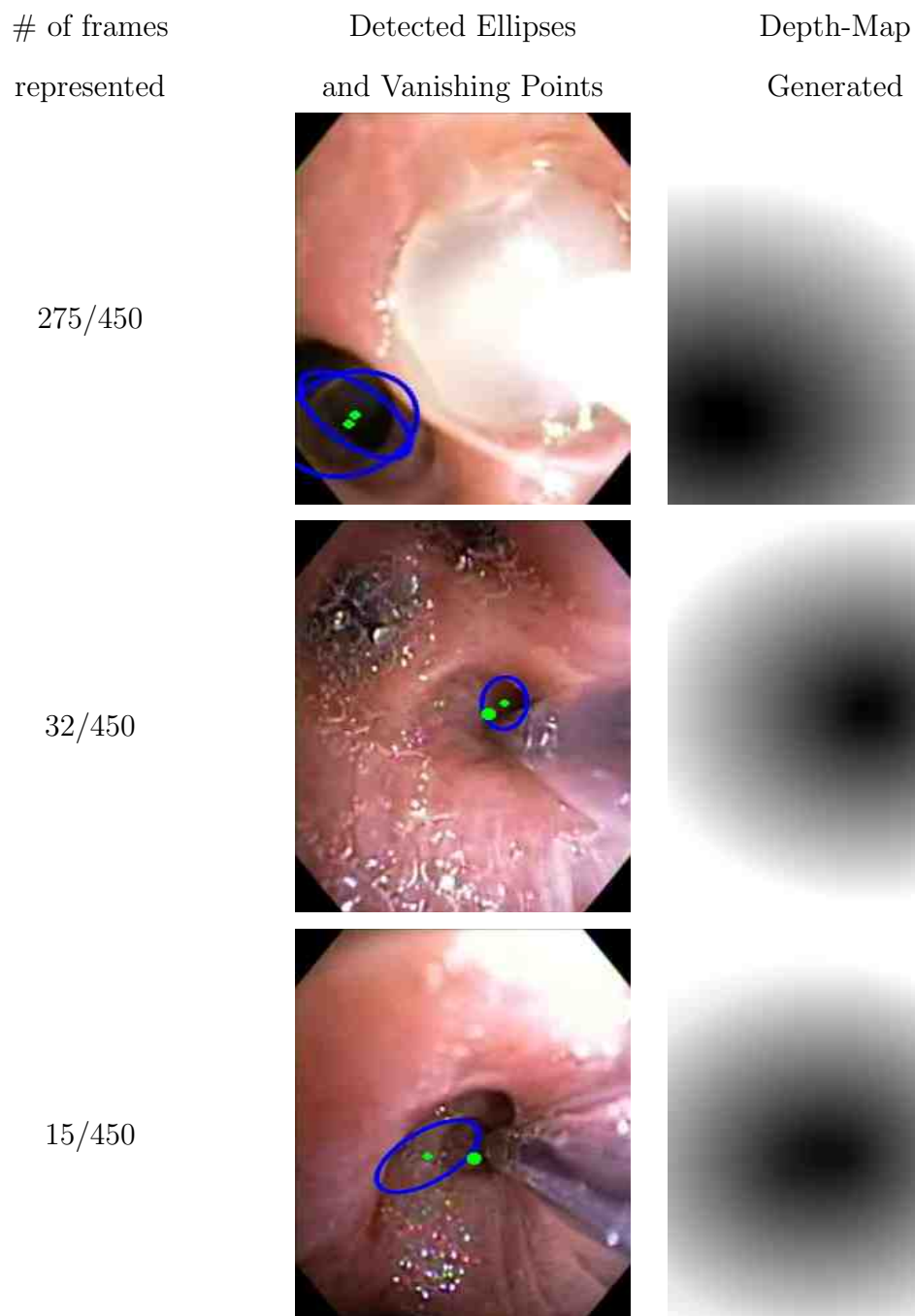


Fig. A.6. Result of algorithm implemented on dataset with one detected vanishing point from ellipse and one inferred (continued from Figure A.5)

B. CODE IN VISUAL C/C++ USING OPENCV LIBRARY

B.1 Functions for Stages 1, 2 and 4

```

    /* File Name: preprocessing_functions.cpp */
#include <cv.h>
#include <highgui.h>
#include <time.h>

    /* Function to crop out the region of interest: Stage 1, 2 and 3 processes */
void regionofinterest(IplImage* image1, IplImage* roi_img)
{
    /* int wid = 215;int ht = 304; CvRect rectAngle = cvRect(203,44,wid,ht); */
    /* Dimensions of region of interest for second procedure */
    int wid = 226; int ht = 213; CvRect rectAngle = cvRect(83,11,wid,ht); /* Dimensions
of region of interest for first procedure */
    IplImage* im = cvCreateImage(cvSize(wid,ht),image1->depth,image1->nChannels);
    cvSetImageROI(image1,rectAngle); /* To set the region of interest in image1 */
    cvCopyImage(image1,roi_img); /*To copy the region of interest of image1 into the
image, roi_image* /
    cvResetImageROI(image1); /*To reset the region of interest in image1 */
}

    /* Function to identify the darkest region in the endoscopy image and creating a
mask: Stage 2 process */
void darkest_point(IplImage* image1, IplImage* image2)
{
    CvMat* imgmat = cvCreateMat(image1->height,image1->width,CV_32FC3);
    cvConvert(image1, imgmat); /* convert input image to matrix form */

```

```

int s = image1->height*image1->width;
CvPoint *p = NULL;p = new CvPoint[s];
for(int k=0; k<s; k++)
{
p[k].x = 0; p[k].y = 0;
}/* initialize cvpoint 'p' for storing image pixels */
int count = 0; /* initialize counter to zero */
for(int i=0; i<imgmat->rows; i++)
{
for(int j=0; j<imgmat->cols; j++)
{
CvScalar scal = cvGet2D(imgmat,i,j); /* access each (i,j)th element of matrix, 'img-
mat' */
int b = (int)scal.val[0]; /* store blue component of image1 in b */
int g = (int)scal.val[1]; /* store green component of image1 in g */
int r = (int)scal.val[2]; /* store red component of image1 in r */
if((b<65)&&(g<50)&&(r<95)) /* if b,g,r component values are each less than a
threshold value */
{
p[count].x = j; p[count].y =i;/* (j,i) are the coordinates of the point in image1 which
has darkest pixel values */
cvCircle(image2,p[count],1,CV_RGB(255,255,255),1,CV_AA,0); /* mask is transparant
in the darkest region */
count++; /*increment counter*/
}
else/* if b,g,r component values are each greater than the threshold value */
{
p[count].x = j; p[count].y =i;/* (j,i) are the coordinates of the point in image1 which
do not have darkest pixel values */
}
}
}
}

```

```

cvCircle(image2,p[count],1,CV_RGB(0,0,0),1,CV_AA,0); /* mask is opaque in such
regions */
count++; /*increment counter*/
}
}
}
}

/* Function to mask one image with another: Stage 2 process */
void showMaskPart(IplImage* image1, IplImage* mask, IplImage* result)
{
/* image1 is the source image which is to be masked
* mask is a single channel binary image as a mask
* result is the image with the same size, depth, channel with src
*/
cvZero(result);
CvSize sz = cvSize(image1->width, image1->height);
IplImage* refImg = cvCreateImage(sz, image1->depth, image1->nChannels);
cvZero(refImg);
cvOr(image1, refImg, result, mask);/* applying OR operation on each pixel of 'im-
age1' and 'mask' and storing result in 'result' */
cvReleaseImage(&refImg);
}

/* Function for stitching image 'src2' to the right side of image 'src1' */ /* and
storing result in image 'dst' */
void stitching(IplImage* src1,IplImage* src2, IplImage* dst)
{
CvRect left = cvRect(0,0,src1->width,dst->height); /* setting region of interest for
src1 image */
cvSetImageROI(dst,left);

```

```

cvCopy(src1,dst,0);
cvResetImageROI(dst); /* reset 'dst' image region of interest */
CvRect right = cvRect(src2->width,0,src2->width,dst->height);/* setting region of
interest for src2 image */
cvSetImageROI(dst,right);
cvCopy(src2,dst,0);
cvResetImageROI(dst); /* reset 'dst' image region of interest */
}
/* Function to find darkest point for inferred vanishing point in case single ellipse
is detected */
void find_one_vanishing_point(IplImage* image1, int& x, int& y)
{
CvMat* imgmat = cvCreateMat(image1->height,image1->width,CV_32FC3);
cvConvert(image1, imgmat); /* converting input image to matrix form */
int s = image1->height*image1->width;
int count = 0; /* counter to count number of pixels in having darkest value */
CvPoint *p = NULL;p = new CvPoint[s];
for(int k=0; k<s; k++)
{
p[k].x = 0; p[k].y = 0; } /* initializing cvpoint 'p' for storing image pixels */
for(int i=0; i<imgmat->rows; i++)
{
for(int j=0; j<imgmat->cols; j++)
{
CvScalar scal = cvGet2D(imgmat,i,j); /* accessing each (i,j)th element of matrix,imgmat
*/
int b = (int)scal.val[0]; /* store blue component of image1 in b */
int g = (int)scal.val[1]; /* store green component of image1 in g */
int r = (int)scal.val[2]; /* store red component of image1 in r */

```

```

/* if b,g,r component values are below a certain threshold,count the point as a pixel
having darkest value */
if((b<15)&&(g<15)&&(r<50))
{
p[count].x = j; p[count].y =i;
count++;
}
}
}
int sumx = 0; int sumy = 0; double meanx,meany;
/* finding mean of all pixels having darkest value */
for(int i=0; i<count; i++)
{
sumx = sumx + p[i].x;
sumy = sumy + p[i].y;
}
meanx = sumx/count;
meany = sumy/count;
x = cvRound(meanx);
y = cvRound(meany);
}

```

B.2 Functions for Stage 3: Ellipse Detection

```

/* File Name: ellipse_detection.cpp */
#include <cv.h>
#include <highgui.h>
#include "ellipse_detection_main.cpp" /* including the file containing main function,

```

```

which is the calling function */
#include <time.h>

/* Function to find element repeating maximum number of times in a 1D array
*/
int maxelement(double num[], int lim, int& maxcount)
{
/* arranging elements of the array, num[] in descending order */
double temp;
for(int i=0; i<lim; i++)
{
for(int j=i+1; j<lim; j++)
{
if(num[i] <= num[j])
{
temp = num[j];
num[j] = num[i];
num[i] = temp;
}
}
}/* now elements of the array, num[] are in descending order */
int currentValue = 0; int currentCount = 0;
int maxValue = 0; int maxCount = 0;
for(int i=0; i<lim; i++) /* scanning through each element of the array, num[] */
{
/* is currentValue equal to the value of current element in the array, num[] */
if(currentValue == cvRound(num[i]))
{
currentCount++; /* if yes, increment currentCount */
}
}

```

```

else /* if currentValue not equal to value of current element in the array, num[] */
{
/* is this value of currentCount greater than max count */
if(currentCount > maxCount)
{
/* if yes */
maxCount = currentCount;
maxValue = currentValue;
}
/* reset values */
currentValue = cvRound(num[i]);
currentCount = 0;
}
}

maxcount = maxCount; /* maxcount is the vote of number of times maxValue repeats
in the array, num[] */
return(maxValue); /* maxValue is the value that repeats maximum number of times
in the array, num[] */
}

/* Function to find median of a 1D array */
int findmedian(int arr[],int lim)
{
if(lim%2==0)
{
return (arr[lim/2]+arr[lim/2-1])/2;
}
else
{
return arr[lim/2];
}
}

```



```

}
}
/* Function for ellipse detection */
void detect_ellipses(IplImage* image1, IplImage* image2, int* majoraxis,
int* minoraxis, int* centersx, int* centersy, double* orientations, int sizeofarray)
{
/* initializations */
double a2dist,a2,ddist,d,a,fdist,f,bdist,b,maxim;
double costau, sintau;
double one,two,slp,alfa;
int ori[15],maja[15],mina[15],centx[15],centy[15],counting = 0;
int count,vote;
CvPoint cent;
for(int k=0; k<sizeofarray; k++)
{
majoraxis[k] = 0; minoraxis[k] = 0;
centersx[k] = 0; centersy[k] = 0; orientations[k] = 0.0;
}
for(int k=0; k<15; k++)
{
maja[k] = 0; mina[k] = 0; centx[k] = 0; centy[k] = 0; ori[k] = 0;
}
/* accumulator matrix */
CvMat* accum = cvCreateMat(mat->rows,mat->cols,CV_32FC1);
/* accumulator array */
double *ac = NULL; ac = new double[mat->rows * mat->cols];
for(int k=0; k<mat->rows * mat->cols; k++)
ac[k] = 0.0;
IplImage* out = cvCloneImage(image2);

```

```

int tot;
counting = 0; /* initializations end */
/* start timer to check execution time of ellipse detection */
clock_t start1 = clock();
CvMat* mat = cvCreateMat(image1->height,image1->width,CV_32FC1);
cvConvert(image1,mat); /* store all edge pixels in a 2D array */
tot = cvCountNonZero(mat); /* counting total number of edge pixels */
printf("Total edge pixels = %d",tot); /* printing total number of edge pixels */
/* for each edge pixel with coordinates (x1,y1) */
for(int x1=0; x1<mat->cols; x1++)
{
for(int y1=0; y1<mat->rows; y1++)
{
if((x1!=0)&&(y1!=0)&&((float*)(mat->data.ptr + mat->step*y1))[x1] == 255)
{
/* for each edge pixel with coordinates (x2,y2) */
for(int y2=0; y2<mat->rows; y2++)
{
for(int x2=0; x2<mat->cols; x2++)
{
if((x2!=0)&&(y2!=0)&&((float*)(mat->data.ptr + mat->step*y2))[x2] == 255)
{
/* length of semi major axis, 'a' is half of distance between points (x1,y1) and (x2,y2)
*/
a2dist = (double)((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1));
a2 = sqrt(a2dist); /* length of major axis */
a = a2/2; /* length of semi major axis from Equation 2.3 */
/*is length of major axis between 8 and 100 units*/
if((a2 >= 8)&&(a2 < 100))

```

```

{
/* if length of major axis is between 8 and 100 units */
/* x and y coordinates of center calculated from Equations 2.1 and 2.2 */
cent.x = cvRound((x1+x2)/2);
cent.y = cvRound((y1+y2)/2);
/* calculating slope, 'slp' and orientation, 'alfa' of ellipse */
one = (double)(y2-y1);
two = (double)(x2-x1);
slp = one/two;
/* (slp < 0) conditions */
if(((x1>x2)&&(y1<y2)) || ((x1<x2)&&(y1>y2)))
{
/* orientation angle of ellipse calculated from modification of Equation 2.4 */
alfa = (-1)*((atan(slp))*(360/(2*CV_PI)));
}
else if(((x1<x2)&&(y1<y2)) || ((x1>x2)&&(y1>y2)))
{
/* orientation angle of ellipse calculated from modification of Equation 2.4 */
alfa = (-1)*(atan(slp)*(360/(2*CV_PI)));
}
/*(slp >= 0) conditions*/
else if(x1!=x2 && y1!=y2)
{
/* orientation angle of ellipse calculated from Equation 2.4 */
alfa = atan(slp) *(360/(2*CV_PI));
}
else if((x1 == x2) && (y1!=y2))
{
alfa = 90;
}
}

```

```

}
else if((y1 == y2) && (x1!=x2))
{
  alfa = 0;
}
count = 0;
cvZero(accum); /* clearing accumulator matrix */
/* for each edge pixel with coordinates (x,y) */
for(int y=0; y<mat->rows; y++)
{
  for(int x=0; x<mat->cols; x++)
  {
    if((x!=0)&&(y!=0)&&((float*)(mat->data.ptr + mat->step*y))[x] == 255)
    {
      if(x!=x1 && x!=x2 && y!=y1 && y!=y2)
      {
        ddist = ((cent.x - x)*(cent.x - x)) + ((cent.y -y)*(cent.y - y));
        d = sqrt(ddist); /* calculating distance between (x0,y0) and (x,y) */
        fdist = ((x2-x)*(x2-x))+((y2-y)*(y2-y));
        f = sqrt(fdist); /* calculating distance between (x2,y2) and (x,y) */
        costau = ((a*a) + (d*d) - (f*f))/(2*a*d); /* calculating cos  $\tau$  from Equation 2.6 */
        if((d<=a) && (costau >= -1) && (costau <= 1) )
        {
          sintau = sqrt(1.0-(costau*costau)); /* calculating sin  $\tau$  value from trigonometric identity */
          if((sintau >= -1) && (sintau <= 1) && (((a*a) - (d*d*costau*costau)) !=0))
          {
            bdist = abs((a*a*d*d*sintau*sintau)/((a*a) - (d*d*costau*costau)));
            b = sqrt(bdist); /* calculating semi minor axis length from Equation 2.5 */

```

```

/* if semi-minor axis length is greater than '8' */
if(cvRound(b)>8)
{
/* store the location of edge pixel corresponding to */
/* this (greater than '8') value of semi-minor axis */
/* in accumulator matrix */
((int*)(accum->data.ptr + accum->step*y))[x] = cvRound(b);
/* increment accumulator array for this value of semi-minor axis */
ac[count] = b;
count++;
} /* if(b) ends */
} /* if(sintau) ends */
} /* if(costau) ends */
} /* if((x,y) != (x1,y1) && (x,y) != (x2,y2)) ends */
} /* if(considering only edge pixels x,y) ends */
} /* for(x) ends */
} /* for(y) ends */

maxim = maxelement(ac,count,vote); /* finding value of 'b' that repeats maximum
number of times in accumulator array */

CvPoint pt;

/* is value of 'vote' satisfying the threshold needed for the detected ellipse to be the
correct ellipse */
if(((tot<=1600)&&(vote > 27)&&(vote<30)) || ((tot>1600)&&(vote>47)))
{
/* if yes, one ellipse is detected */
/* detected ellipse with the corresponding parameter values is superimposed on the
source image */
CvSize axes;
axes.height = cvRound(maxim); /* minor axis length */

```

```

axes.width = cvRound(a); /* major axis length */
/* store parameter values of detected ellipse in 1D arrays */
maja[counting] = axes.width; mina[counting] = axes.height;
centx[counting] = cent.x; centy[counting] = cent.y; ori[counting] = cvRound(alfa);
/* draw ellipse with detected parameters on source image */
cvEllipse(image2,cent,axes,alfa,0,360,CV_RGB(255,0,0),1,CV_AA,0);
counting++; /* count number of ellipses detected */
/* deleting edge pixels that belong to the detected ellipse */
/* for each edge pixel with coordinates (x1,y1) */
for(int yi=0; yi<mat->rows; yi++)
{
for(int xi=0; xi<mat->cols; xi++)
{
/* is value in accumulator matrix at point (x1,y1) equal to value of minor axis of
detected ellipse */
if((xi!=x1)&&(yi!=y1)&&
((int*)(accum->data.ptr + accum->step*yi))[xi] == cvRound(maxim))
{
/* if yes, make the value in edge image matrix at point (x1,y1) equal to zero */
/* this deletes the points in edge image matrix that lie on detected ellipse */
pt.x = xi; pt.y = yi;
((int*)(mat->data.ptr + mat->step*yi))[xi] = 0;
} /* if(maxim) ends */
} /* for(j) ends */
} /* for(i) ends */
} /*if(vote) ends */
cvZero(accum); /* clearing accumulator array */
} /* if(a2) ends */
} /* if(considering only edge pixels x2,y2) ends */

```

```

} /* for(x2) ends */
} /* for(y2) ends */
} /* if(considering only edge pixels x1,y1) ends */
} /* for(x1) ends */
} /* for(y1) ends */

/* Printing time taken for all ellipses to be detected in edge image */
printf("Total time elapsed for ellipse detection: %f", ((double)clock() - start1) /
CLOCKS_PER_SEC);

/* initializations for grouping detected ellipses into two groups */
int centxsum[2] = {0,0}, centysum[2] = {0,0};
int majorsum[2] = {0,0}, minorsum[2] = {0,0};
int anglesum[2] = {0,0}, k0=0, k1=0;
int medx,medy,xcent,ycent;
double distmedsq,distmed; /* initializations for grouping ellipses end */

/* 'counting' denotes total number of ellipses detected in the given source image */
/* if number of ellipses detected is even */
if(counting%2==0)
{
/* find median of array containing x and y coordinates of centers of the detected
ellipses with array size as 'counting+1' */
medx = findmedian(centx,counting+1);
medy = findmedian(centy,counting+1);
}
else if(counting%2==1) /* if number of ellipses detected is odd */
{
/* find median of array containing x and y coordinates of centers of the detected
ellipses with array size as 'counting' */
medx = findmedian(centx,counting);
medy = findmedian(centy,counting);
}

```

```

}
/* for each element in array containing (x0,y0) of detected ellipses */
for(int i=0; i<=counting; i++)
{
/* making sure elements of the arrays are not zero */
if((centx[i] != 0)&&(centy[i] != 0))
{
xcent = centx[i]; ycent = centy[i];
/* finding distance between median of (x0,y0) and current (x0,y0) point */
distmedsq = (double)(((medx-xcent)*(medx-xcent)) + ((medy-ycent)*(medy-ycent)));
distmed = sqrt(distmedsq);
/* if distance between median and current (x0,y0) value is less than 35 */
if((distmed < 35)&&(distmed >= 0))
{
/* add parameter values of ellipses with centers near each other and store as group
one*/
centxsum[0] = centxsum[0] + xcent; centysum[0] = centysum[0] + ycent;
anglesum[0] = anglesum[0] + ori[i];
majorsum[0] = majorsum[0] + maja[i]; minorsum[0] = minorsum[0] + mina[i];
k0++; /* counting number of ellipses in group one */
}
else
{
/* else, add parameter values of remaining ellipses and store as group two*/
centxsum[1] = centxsum[1] + xcent; centysum[1] = centysum[1] + ycent;
anglesum[1] = anglesum[1] + ori[i];
majorsum[1] = majorsum[1] + maja[i]; minorsum[1] = minorsum[1] + mina[i];
k1++; /* counting number of ellipses in group two */
}
}

```



```

}
}
/* finding mean of parameters in groups one and two */
/* storing result as output parameters of the function, detect_ellipses() */
if(k0!=0)
{
centersx[0] = cvRound(centxsum[0])/k0; centersy[0] = cvRound(centysum[0])/k0;
orientations[0] = anglesum[0]/k0;
majoraxis[0] = cvRound(majorsum[0])/k0; minoraxis[0] = cvRound(minorsum[0])/k0;
}
/* if there are no ellipses in group one, store the output parameter values as zero */
else if(k0==0)
{
centersx[0] = 0; centersy[0] = 0; orientations[0] = 0;
majoraxis[0] = 0; minoraxis[0] = 0;
}
if(k1!=0)
{
centersx[1] = cvRound(centxsum[1])/k1; centersy[1] = cvRound(centysum[1])/k1;
orientations[1] = anglesum[1]/k1;
majoraxis[1] = cvRound(majorsum[1])/k1; minoraxis[1] = cvRound(minorsum[1])/k1;
}
/* if there are no ellipses in group two, store the output parameter values as zero */
else if(k1==0)
{
centersx[1] = 0; centersy[1] = 0; orientations[1] = 0;
majoraxis[1] = 0; minoraxis[1] = 0;
}
}
}

```

B.3 Main Functon

```

    /* File Name: ellipse_detection_main.cpp */
#include <iostream>
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <direct.h>
#include <errno.h>
using namespace std;
using namespace cv;
void regionofinterest(IplImage* image1, IplImage* roi_img);
void showMaskPart(IplImage* image1, IplImage* mask, IplImage* result);
void detect_ellipses(IplImage* image1, IplImage* image2, int* majoraxis,
int* minoraxis, int* centersx, int* centersy, double* orientations, int sizeofarray);
void darkest_point(IplImage* image1, IplImage* image2);
void stitching(IplImage* src1,IplImage* src2, IplImage* dst);
/* global variables */
IplImage* src; IplImage* roiimg;
IplImage* depthmap;
/* Main function */
int main(int argc, char* argv[])
{
/* initializations for ellipse detection */
int a[2],b[2],cx[2],cy[2],circx,circy;
double o[2];
for(int i=0; i<2; i++)

```

```

{
a[i] = 0;b[i] = 0;cx[i] = 0;cy[i] = 0;o[i] = 0.0;
}
circx = 0; circy = 0;
/* loading source image */
if (argc < 2)
{
fprintf(stderr, "Usage: %s <image>", argv[0]);
return 1;
}
int c=1;
/* for 61 images given as input arguments. this value can be changed according to
desired number of images to be given as input arguments */
while(c <= 61)
{
src = cvLoadImage(argv[c], 1);
depthmap = cvCreateImage(cvGetSize(src),src->depth,1);
cvCvtColor(src,depthmap,CV_BGR2GRAY);
/* calling the preprocessing functions */
/* finding region of interest */
/* int width = 254; int height = 361; */ /* for 2nd procedure, part 2 */
/* int width = 215; int height = 304; */ /* for 2nd procedure, part 1 */
int width = 226; int height = 213; /* for 1st procedure */
CvRect rectAngle = cvRect(83,11,width,height);
roiimg = cvCreateImage(cvSize(width,height),src->depth,src->nChannels);
cvSetImageROI(src,rectAngle);
cvCopyImage(src,roiimg);
cvResetImageROI(src);
cvSaveImage("roi.png",roiimg,0); /* saving region of interest image as 'roi.png' */

```

```

/* median X filtering */
IplImage* median = cvCreateImage(cvGetSize(roiimg),roiimg->depth, 3);
cvSmooth(roiimg,median,CV_MEDIAN,5,0);
cvSaveImage("roimed.png",median); /* saving median filtered image as 'roimed.png'
*/
/* darkest area detection */
IplImage* dark = cvCloneImage(median);
darkest_point(median, dark);
/* dilate darkest area detection image to create a mask */
IplImage* dark_dil = cvCreateImage(cvGetSize(dark),dark->depth,dark->nChannels);
cvDilate(dark,dark_dil,NULL,2); /* performing two iterations of dilation */
cvSaveImage("mask.png",dark_dil,0); /* saving mask image as 'mask.png' */
/* convert 'dark_dil' to grayscale for edge detection */
IplImage* dark_dil_gray = cvCreateImage(cvGetSize(dark_dil),dark_dil->depth,1);
cvCvtColor(dark_dil,dark_dil_gray,CV_RGB2GRAY);
/* masking median filtered image with mask and store result in 'masked' */
IplImage* masked = cvCreateImage(cvGetSize(median),median->depth,median->nChannels);
showMaskPart(median,dark_dil_gray,masked);
cvSaveImage("masked.png",masked); /* saving masked image as 'masked.png' */
/* convert masked image to grayscale */
IplImage* mask_gray = cvCreateImage(cvGetSize(masked),masked->depth,1);
cvCvtColor(masked,mask_gray,CV_RGB2GRAY);
/* canny edge detection */
IplImage* edge = cvCreateImage(cvGetSize(roiimg),roiimg->depth, 1);
cvCanny(mask_gray,edge,100,70,3);
cvSaveImage("edge.png",edge,0); /* saving canny edge detected image as 'edge.png'
*/
/* ellipse detection */
IplImage* ell = cvCloneImage(masked);

```

```

IplImage* ell1 = cvCloneImage(roiimg);
IplImage* ell2 = cvCloneImage(roiimg);
int size = 2;
detect_ellipses(edge,ell,a,b,cx,cy,o,size); /* detect ellipse on edge segmented image */
/* depth map generation */
IplImage* depthimg = cvCreateImage(cvGetSize(ell),ell->depth, 1);
cvSetImageROI(depthmap,rectAngle);
cvCopyImage(depthmap,depthimg); /* creating image in which the depth map will
be stored */
/* creating images in which the depth map will be stored */
IplImage* depthimg1 = cvCreateImage(cvGetSize(ell),ell->depth, 1);
IplImage* depthimg2 = cvCreateImage(cvGetSize(ell),ell->depth, 1);
/* initializations */
cvZero(depthimg);
cvZero(depthmap);
cvZero(depthimg1);
cvZero(depthimg2);
CvPoint pt;
pt.x = 0;
pt.y = 0;
CvSize ax; CvPoint cen; double angl;
for(int k=0; k< size; k++)
{
/* if no ellipses are detected after ellipse detection function */
/* depth gradient generation */
if((a[k]==0)||(b[k]==0)||(cx[k]==0)||(cy[k]==0))
{
find_one_vanishing_point(median,circx,circy);
pt.x = circx; pt.y = circy;

```

```

if(k == 0)
{
/* growing circles of varying radii and color(grayscale) in the 'depthmap1' image */
for(int i = ell->width+ell->height; i>0; i--)
{
int j=3*i/2;
cvCircle(depthimg1, pt, i, CV_RGB(j,j,j), 5, CV_AA, 0);
}
}
else if(k == 1)
{
/* growing circles of varying radii and color(grayscale) in the 'depthmap2' image */
for(int i = ell->width+ell->height; i>0; i--)
{
int j=3*i/2;
cvCircle(depthimg2, pt, i, CV_RGB(j,j,j), 5, CV_AA, 0);
}
}
}
/* else, if ellipse is detected after ellipse detection function */
else
{
ax.height = b[k]; ax.width = a[k]; cen.x = cx[k]; cen.y = cy[k]; ang1 = o[k];
if(k==0)
{
CvSize ax1;
/* drawing depthimg1 gradient */
/* growing ellipses of varying axes lengths and color(grayscale) in the 'depthmap1'
image */

```

```

for(int i = depthimg1->width+depthimg1->height; i>0; i-)
{
ax1.height = (b[k]) + i;
ax1.width = (a[k]) + i;
int j=3*i/2;
cvEllipse(depthimg1,cen,ax1,ang1,0,360,CV_RGB(j,j,j),5,CV_AA,0);
}
} /* if (k=0) ends */
if(k==1)
{
CvSize ax2;
/* drawing depthimg2 gradient */
/* growing ellipses of varying axes lengths and color( grayscale) in the 'depthmap2'
image */
for(int i = depthimg2->width+depthimg2->height; i>0; i-)
{
ax2.height = (b[k]) + i;
ax2.width = (a[k]) + i;
int j=3*i/2;
cvEllipse(depthimg2,cen,ax2,ang1,0,360,CV_RGB(j,j,j),5,CV_AA,0);
}
} /* if (k=1) ends */
}
} /* for (k) ends */
/* averaging depthimg1 and depthimg2 images and storing the result in 'depthimg'
*/
cvAddWeighted(depthimg1, 1./2., depthimg2, 1./2., 0.0, depthimg );
cvCopy(depthimg,depthmap,NULL); /* copying 'depthimg' into 'depthmap' */
/* resetting image regions of interest */

```

```
cvResetImageROI(depthmap);  
/* stitching src image and depthmap image beside each other and saving result in  
'stitched'*/  
int wid = depthmap->width + src->width;  
int ht = src->height;  
IplImage* stitched = cvCreateImage(cvSize(wid,ht),depthmap->depth,3);  
IplImage* depthmap_color = cvCreateImage(cvGetSize(depthmap),depthmap->depth,3);  
cvCvtColor(depthmap,depthmap_color,CV_GRAY2BGR); /* converting grayscale im-  
age to color image */  
stitching(src,depthmap_color,stitched);  
c++;  
} /* while (c) ends */  
/* cleaning up */  
cvDestroyAllWindows();  
return 0;  
}
```