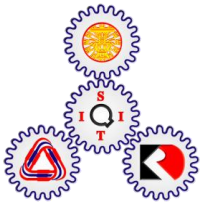


CORNER POINT DETECTION VIA WALKING PARTICLES

BY

PASINDU MANISHA KURUPPUARACHCHI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ACADEMIC YEAR 2018

CORNER POINT DETECTION VIA WALKING PARTICLES

BY

PASINDU MANISHA KURUPPUARACHCHI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2018

CORNER POINT DETECTION VIA WALKING PARTICLES

A Thesis Presented

By

PASINDU MANISHA KURUPPUARACHCHI

Submitted to

Sirindhorn International Institute of Technology

Thammasat University

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE (ENGINEERING AND TECHNOLOGY)

Approved as to style and content by

Advisor

(Prof. Dr. Stanislav S. Makhanov)

Committee Member and
Chairperson of Examination Committee

(Asst. Prof. Dr. Annupan Rodtook)

Committee Member

(Asst. Prof. Dr. Pakinee Aimmanee)

NOVEMBER 2018

Acknowledgements

Doing a research is not a road with roses. There are many obstacles and roadblocks to overcome. Conducting a research alone is impossible. This is my gratitude to all of the people helps me throughout this journey.

First of all, I would like to express my gratitude to the SIIT for selecting me as a scholarship graduate student. This was a one of kind experience and I learned a lot academically as well as life skills throughout this amazing two and a half years in Thailand. I was exposed to many extracurricular activities and workshops with other universities. These programs also helped me indirect way to shape my research and gain new skills.

The Academic advisor play vital role in every research and that person is always behind the success of the research. Prof. Dr. Stanislav S. Makhanov helps me throughout this research journey and always there to helps me with all the necessary support and guidance. I would like to thank my committee for giving out kind advisors and feedback to improve my research. Committee chairperson Asst. Prof. Dr. Annupan Rodtook and committee member Asst. Prof. Dr. Pakinee Aimmanee input many valuable ideas and suggestions to shape the research.

I would also like to thank all my friends and family to being with me through tuff times and motivate me to conduct my research. Since I am thousands miles away from my country friends helps me to relax my mind and enjoy my stay.

Thank you all the staff members in SIIT and all others who helps me in various ways.

Abstract

CORNER POINT DETECTION VIA WALKING PARTICLES

By

PASINDU MANISHA KURUPPUARACHCHI

Bachelor of Science (Hons) in Information Technology Specialized in computer systems and networking, SLIIT (Sri Lanka), 2012,
Master of Science (Engineering and Technology), Sirindhorn International Institute of Technology, Thammasat University, 2018

We propose a novel method for corner detection based on walking particles (WP) moving around the edge map of the image. The WP follows the edges and are intelligent to avoid the obstacles. The WP records its movement in the database to analyze and get more insights about the image. The main idea of the new method is that at the corner, the particle has less possible options to move, compared with the flat edge. In other terms corner will limit the particles free movement. Lesser the freedom to move higher the chance for be a corner.
To evaluate WP method's success we performed two experiments. Firstly tested on a set of synthetic and real images against the most popular Harris (H) and Shi-Tomasi (ST) corner detection methods. In this synthetic image test, many distortion methods and added noise are introduce to the image to check the robustness of the algorithm. This experiment shows that the proposed method outperforms H and ST methods on 70% of distorted and normal images experiments and all the test cases with noise added images and real image of the evaluation. A Second experiment about retinal image registration and evaluated based on the root mean square error (RMSE). The result depicts that 81% of the test cases record the minimum RMSE by the proposed method. Therefore, the proposed method provides a novel approach towards corner detection.

Keywords: corner detection; image processing, computer vision, particle method, image registration

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Corner detection is a fundamental problem of image processing with applications to image matching, motion tracking, image registration, stereo vision, etc. [1-3]. Corner detection is a fundamental problem of image processing with applications to image matching, motion tracking, image registration, stereo vision, etc. [1-3]. The corner point detection algorithms can be categorized as the intensity based, contour based, and model based methods. The intensity based algorithms search for large intensity variations. The higher the intensity variation, higher the possibility of being a corner. The contour based methods evaluate the edge map and extract contours. Then, the algorithms evaluate the contour's features such as the curvature, inflection points, etc. to detect the corners [3]. The model (template) based algorithms define a moving window and match it with the corner template [4-5]. Unfortunately, intensity based methods are not always robust for blurred or noisy images [1-2]. For some applications the contour based methods produce a minimal number of false positives with respect to other two methods [5]. The model based detectors are fast, however, they are noise-sensitive and do not work well with the images characterized as a textured background [5].

1.1 Statement Of Problem

The main problem with currently available corner point detectors is depend on the threshold. Threshold is the value that need to initialized before the execution and depend on the value results will be different. Assign the correct threshold values are very important and it also vary from image to image. This is a problem in automation because setting those threshold values manually is time consuming and error prone activity.

1.2 Purpose Of Study

The main objective of this research is about find the corner points in an image effectively and accurately without having threshold parameters. These thresholds will limit the possibilities of the algorithm and hard to scale to real world applications.

Main goals of this project.

1) An automated novel corner detection method with higher accuracy and without initializing thresholds.
2) Introduced new corner detection method real world application performance evaluation.

1.3 Thesis Structure

The structure of the thesis is organized as follows:

Chapter 2 Corner detection background knowledge and introduction about comparing methods that are going to use in the evaluation.

Chapter 3 Particle method algorithm components and how it is executing.

Chapter 4 Evaluation of the particle method

Chapter 5 Results and discussion

Chapter 6 Conclusion and future work

Chapter 2
Background

This chapter describe the history of the corner point detection and comparing methods information.

2.1 What is a corner

Corner is a point that where two edges merge each other. Depend on the angle of this convergence corner point area is calculated. Corner points are one of the interesting represent in the images. In corner points there can be more valuable information is hidden. Such as intensity variations, color difference or even specific regions. This is the reason corner points are also known as feature points.

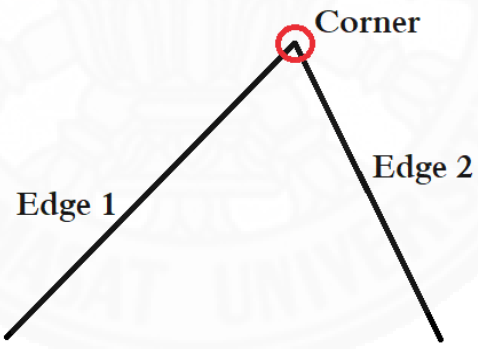
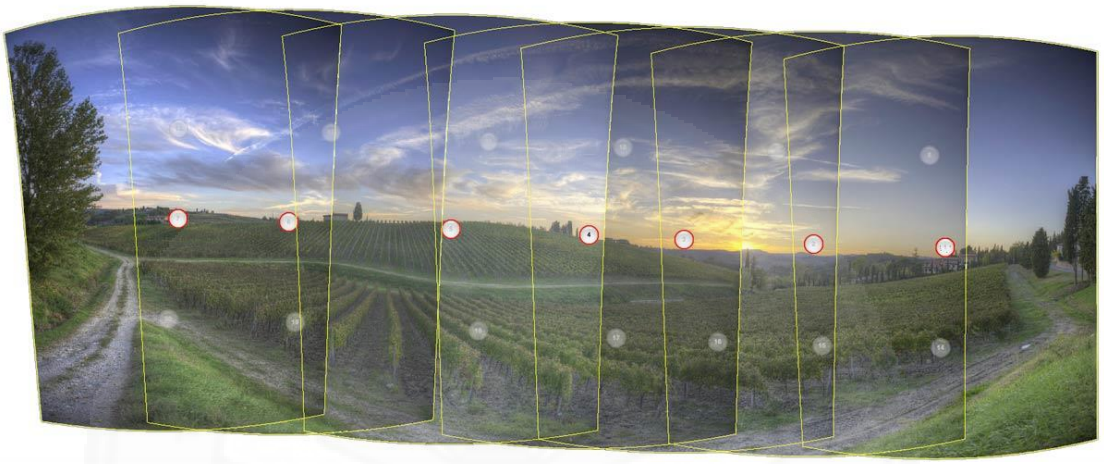


Figure 2.1. Definition of a corner

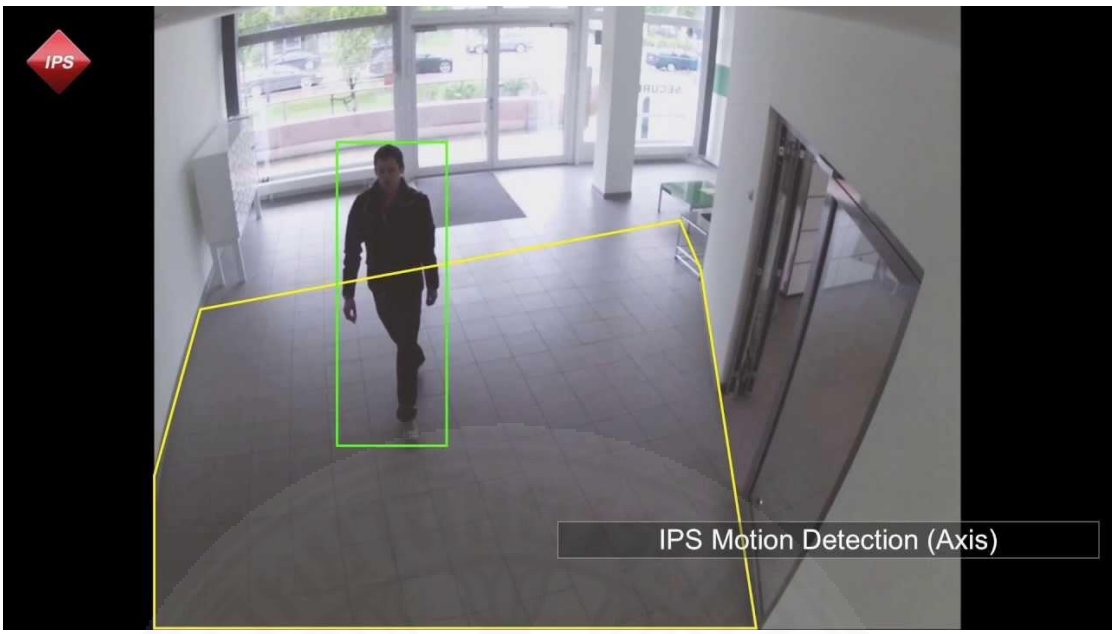
Corner detection is a fundamental problem of image processing with applications to image matching, motion tracking, image registration, stereo vision, etc. [1-3]. In all of these applications, after preprocessing the images (enhance image colors, contrast etc.) next step is detect corners. Corners are the foundation for the many algorithms in these applications.

As an example let's consider, panorama stitching. Panorama is a collection of images take in a continuous manner to create a wider view image. In this use case, those continuous images will be process to detect corners. Based on those corners image overlaps will be detected in order to stitch together. This process will be continue to all the images and final single panorama image will be generated.



(a)

(b)



(c)

Figure 2.2. Applications of corner point detection (a) Panorama stitching (b) Image registration (c) Object detection

The corner point detection algorithms can be categorized as the intensity based, contour based, and model based methods. The intensity based algorithms search for large intensity variations. The higher the intensity variation, higher the possibility of being a corner. The contour based methods evaluate the edge map and extract contours. Then, the algorithms evaluate the contour's features such as the curvature, inflection points, etc. to detect the corners [3]. The model (template) based algorithms define a moving window and match it with the corner template [4-5]. Unfortunately, intensity based methods are not always robust for blurred or noisy images [1-2]. For some applications the contour based methods produce a minimal number of false positives with respect to other two methods [5]. The model based detectors are fast, however, they are noise-sensitive and do not work well with the images characterized as a textured background [5].

2.2 Corner detection methods

For the comparison and evaluate the new corner point method, Harris and Shi-Tomasi corner detection methods are used. Even though (H) and (ST) based on intensity, these two methods are widely used extractors and often used as comparing methods [6]. However, given the variety of the image processing problems, the most suitable corner detection method is still an open problem [22].

Moravec corner detector (1980)

We should easily recognize the point by looking through a small window. Shifting a window in any direction should give a large change in intensity. As shown in the figure 3, moving the window in any direction will give the intensity variation. In the flat regions it was constant and no changes at all. When it comes to edge we can see a symmetric intensity variation. The window will get highest intensity variation around the corner point. This is the basic logic to detect corner in Moravec corner detector.

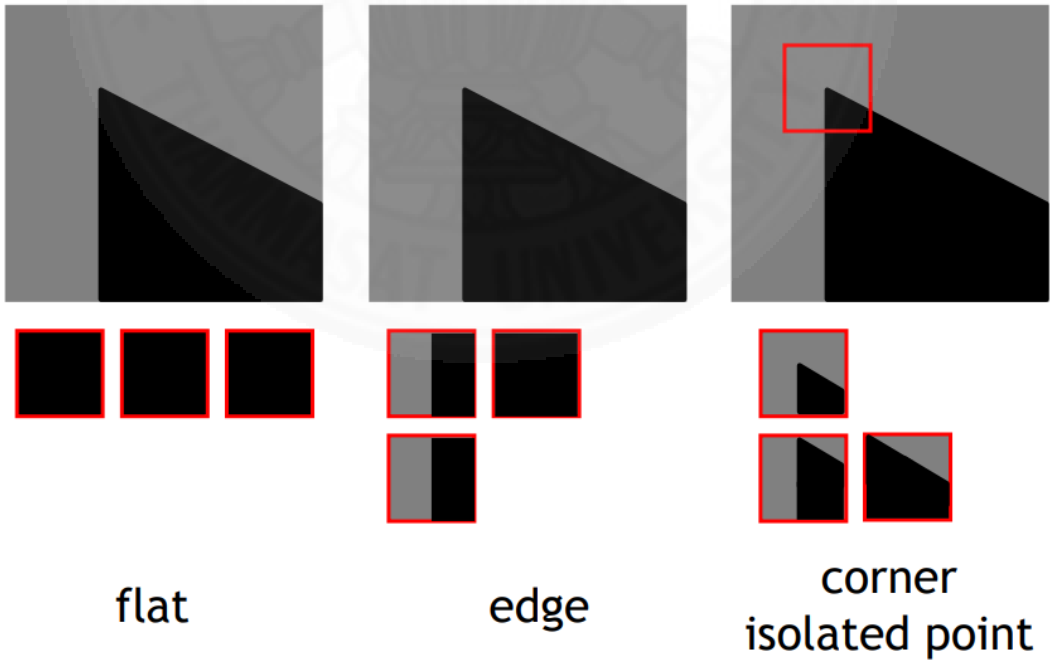


Figure 2.3. Changes of the window to detect flat, edge and corner

Change of intensity for the shift [u, v]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

Intensity

Window function $w(x,y)$ =

1 in window, 0 outside

Figure 2.4. Moravec corner detector intensity calculation formula

The problems with the Moravec corner detector are

- Noisy response due to a binary window function
- Only a set of shifts at every 45 degree is considered
- Responds too strong for edges because only minimum of E is taken into account

Harris corner detector in (1988) solves these problems. In Harris corner detector only a set of shifts at every 45 degree is considered. Consider all small shifts by Taylor's expansion.

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$
$$= \sum_{x,y} w(x,y)\left[I_x u + I_y v + O(u^2, v^2)\right]^2$$

$$E(u,v) = Au^2 + 2Cuv + Bv^2$$
$$A = \sum_{x,y} w(x,y) I_x^2(x,y)$$
$$B = \sum_{x,y} w(x,y) I_y^2(x,y)$$
$$C = \sum_{x,y} w(x,y) I_x(x,y) I_y(x,y)$$

Figure 2.5. Taylor's expansion

Equivalently, for small shifts [u, v] we have a bilinear approximation: where M is a 2×2 matrix computed from image derivatives (1)

The Harris method (HM) is one of the most popular intensity based moving window methods. The basic idea is the detection of significant changes in the window. They are

detected by the eigenvalues of a second moment matrix (1). Where the gray level and subscripts is denoting the partial derivatives [6].

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \quad (1)$$

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2) \quad (2)$$

$$R = \min(\lambda_1, \lambda_2) \quad (3)$$

The score of each point is evaluated by (2), where is the training parameter. The H-detector is translation, rotation and illumination invariant. This method was used with a different degree of success for detecting L type intersections and points with a high curvature along with the potential corner points [4].

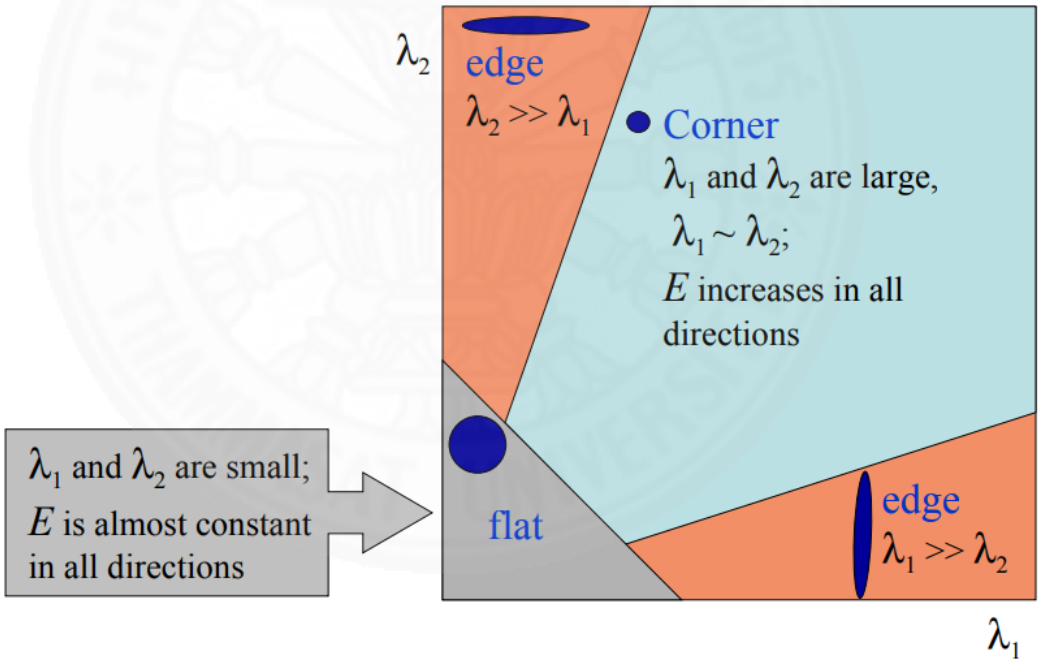


Figure 2.6. Classification of image points using eigenvalues of M

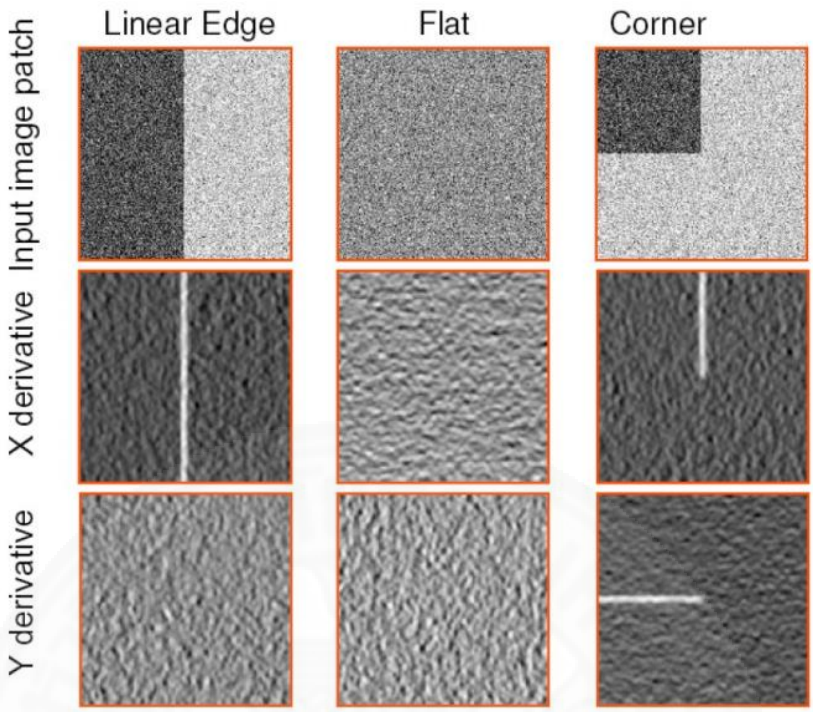


Figure 2.7. Another view of how corners are visible in the window

HM is generally used for camera calibration because of its stability on rotation, viewing angle and luminance [7]. The main drawback of the method, is the high responsiveness to noise [4] [7]. Shi-Tomasi method (STM) is a modification of the HM [8]. A slight variation in the selection criteria, i.e. (3) improves the detector substantially by setting up a minimum quality for the corners. It works well when the H-detector fails to accommodate affine transformations [8].

Note that both HM and STM require a threshold. This often presents a problem in real time applications. Therefore, we propose a novel robust, threshold independent corner detection method based on the WP.

Chapter 3
The Walking Particle Method

This chapter presents the framework and algorithm for the walking particle method. Several important components are there in the algorithm and these components are describe in this chapter.

3.1 A walking particle

The particle is defined as a single unit that move from one pixel at a time and senses, its neighboring pixels using a nine pixel window as shown in figure 7.

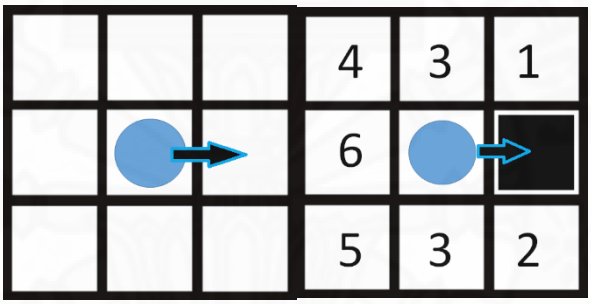


Figure 3.1 WP method 9 pixel window and movement priority for particle going from left to right, black box indicates the obstacle that particle trying to avoid.

Particles are equipped with a data storage unit to store its movements for later processing and decision making based on the particle objective. Particle objective will depend on the side that particle is initialized. Based on this initialization location, the particle velocity is set. The particle movement can be from right to left, left to right, top to bottom or bottom to top. If it detects an obstacle, it will take the decision based on next available position to achieve its objective. As an example, let's consider a particle moving from left to right side of the image boundary. If it detects an obstacle, the particle will try its best to keep moving in the same direction as shown in the figure 1. The particle will first try to move up-right or down-right location. Because in that sense, it's still trying to move to its objective direction, right side. If those two options are not feasible, then it has to choose either to move up or down. Before taking a random

decision, the particle will process the number of free possible movements, available and then take the highest free possible movement direction. Furthermore, it supports to achieve maximum coverage through the image. Then it takes the decision based on that information. If the movement is improbable, the particle should move towards left in order to avoid the obstacle. Since it is undesirable, the particle searches to move up-left position or down-left position. If it is impossible, particle terminates the movement, since moving towards the previous position conflicts its objective.

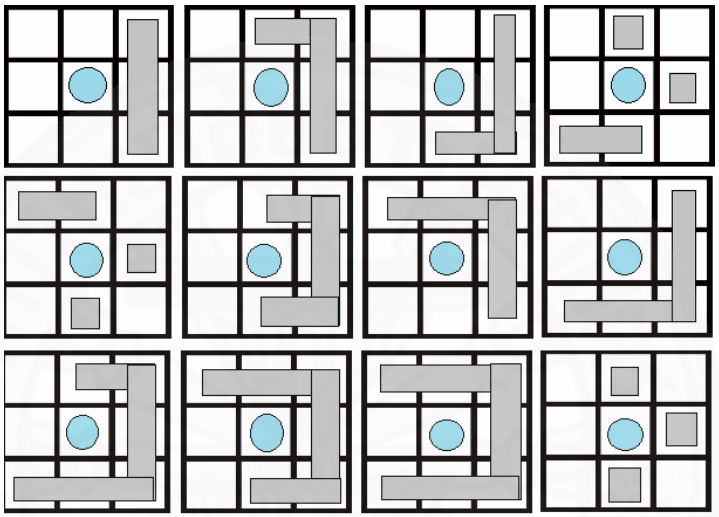


Figure 3.2. Templates of the corners, circle- particle, and rectangle- edge pixel

3.2 Corner detection logic

Based on these movement decisions we can identify corners. If the particle has to move opposite to its objective direction, there is a higher chance of that point being a corner, since more than five locations are blocked in 9 pixel window. There also some other templates that algorithm categories as a corner. It can be observed from the templates in figure 2. These templates are the reason behind the categorization of WPM as a hybrid of contour based and model based corner point detection method. Basic corner point detection logic in WPM is when the particle comes near a corner, it has limited free pixels to move. Most of its pixels in the 9 pixel window are blocked.

3.3 Operation sequence

WPM consists of six modules to detect the corners of a given image. The figure 3 shows the proceedings of these modules. A detailed description about these six modules (fifth and sixth modules are combined) are presented in this section.

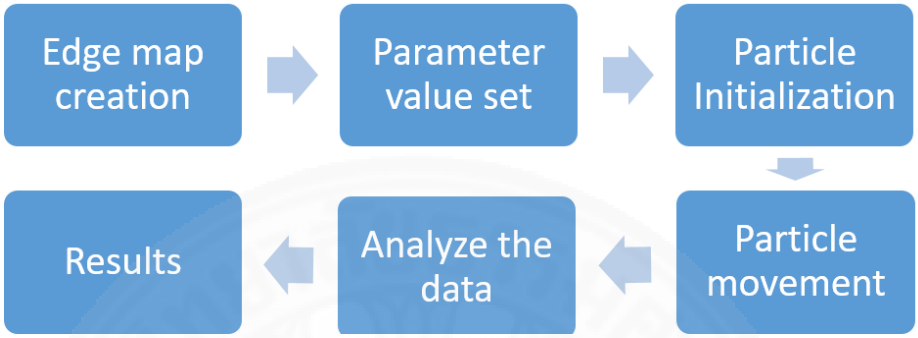


Figure 3.3. WP method corner points detection sequence chart

3.3.1 Edge map creation

WPM use Canny edge detection [5] method to create the edge map as one of the preprocessing technique of the image. One of the main problems that we identify from current corner detection algorithms is the requirement of specifying the threshold value to detect corners correctly. Proposed WPM is a threshold-less corner detection method. In order to create an automated edge map, lower level and upper level of threshold is required. To achieve automatic edge detection, first step is to calculate the median of the single channel pixel intensities. An optional argument, “sigma” can be used to vary the percentage thresholds that are determined based on simple statistics. Thresholds are constructed based on the +/- percentages controlled by the sigma argument. A lower value of sigma indicates a tighter threshold, whereas a larger value gives a wider threshold. In general, it’s not required to change this sigma value often. Default sigma value of 0.33 gives notable results of creating edge map [9]. The Figure 4 shows how automatic edge detection accuracy over lower and higher thresholds. Left most edge map is the low threshold value edge detection while middle edge map is the higher threshold value edge map detection and rightmost one is the automatic threshold value edge map.

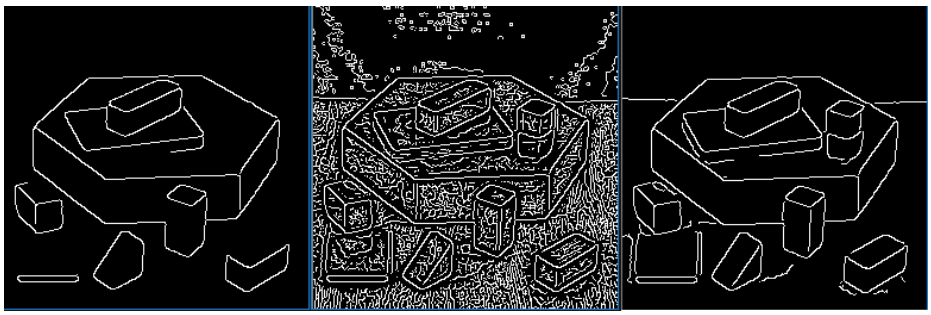


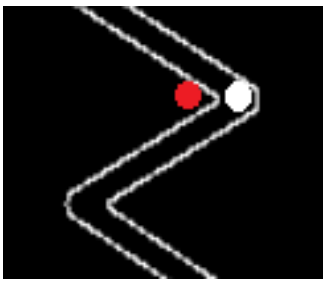
Figure 3.4. Automated edge map detection (low, high and automatic edge detection)

Figure 10. Proved that automatic threshold value edge detection provides less noise and covers a significant amount of edges in the image. Creating a reasonable edge map is a critical part of WPM because if an edge map created with higher noise levels, unwanted and false corner points may detected.

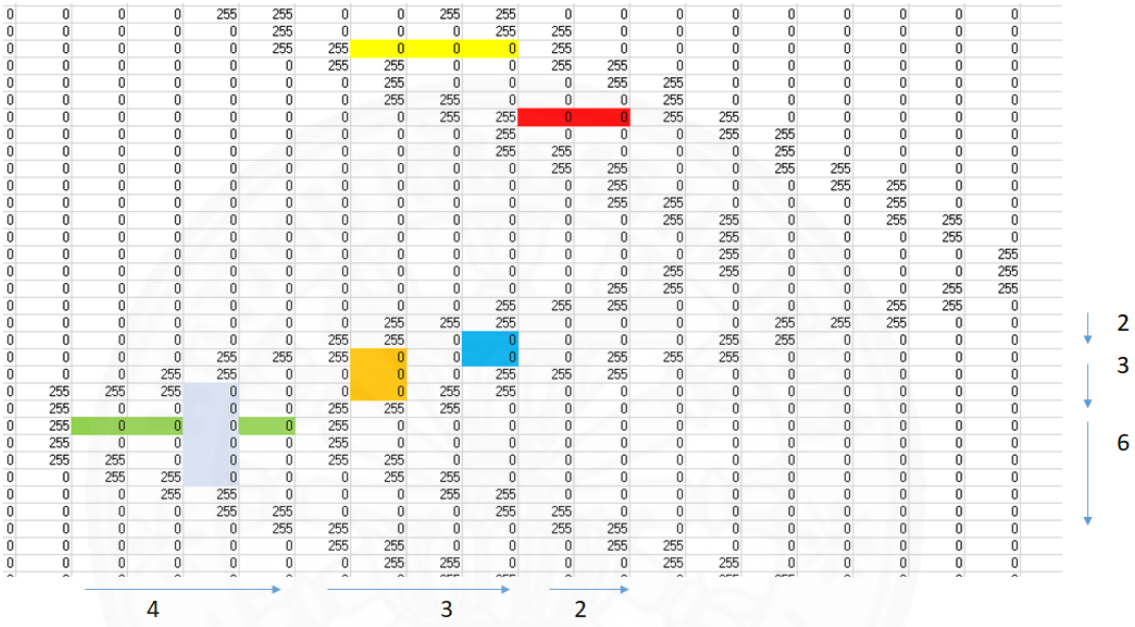
3.3.2 Parameter value set

After creating the edge map of the image, it is required to set initial parameters to operate the algorithm. All these parameters are set automatically. In section 3.2 of the corner point detection logic, it is explained that detecting a corner point means limited possible movement to the particle around the corner. Free movement of a point can be different based on the image. Some images may have more freedom to move but in some case it won't. In order to detect the width of the two edges that create a corner point, it needs to calculate this width parameter. As it is on figure 11 (a), if considering two corner points red and white. The Red corner point is more to expose the outside, but white point is between two edges. In order to detect that white corner point we need to know the distance between two edges.

As shown in Figure 11, the algorithm will take random sampling points based on the size of the image. Then, from these random sample points the particles are sent both horizontal and vertical directions. First, the particle moves until it is able to detect an edge. Then it starts to count the number of pixels that can travel until it senses the next edge.



(a)



(b)

Figure 3.5. (a) Edge map of the selected area (b) Vertical and horizontal random sampling points with its edge to edge distance

In the example of widths and heights of a given edge map path is shown in Fig. 11 (b). For horizontal samples, 4, 3 and 2 pixels widths are detected while for vertical samples 2, 3 and 6 pixels heights detected. In this WP algorithm we choose the smallest value from both vertical and horizontal directions as the free movement parameter. In this scenario, the free movement value is less than 2 pixels. If the free movement of a particle at any given point is less than 2 pixels for at least three directions, the point is detected as a corner.

3.3.3 Particle Initialization

The particle initialization is also one of the important components in this algorithm. There are two objects to be completed in this part, which are achieving maximum

coverage throughout the image and keeping the total particle count as minimum as plausible to a level without burdening the performance. These objectives are achieved by initializing particles from all four directions, dividing the image in three subsections, and initializing the particles repetitively from those three sections. First initializing positions will be the four boundaries of the image and then the image is divided into three sections to achieve maximum coverage. From these boundaries the algorithm tries to detect edges and record the number of initialization points. In figure 12 shows how these imaginary boundaries detect the initialized points. When the algorithm detects a possible initialization point, it will send a particle one pixel above or below to the initialization points detected by vertical orientation boundaries. The points detected by horizontal orientation boundaries will have one pixel right and left. Figure 12 shows how these particle initializing will be detected. There is a reason to initialize two particles from one initialization point. Figure 13 shows all the possible scenarios of initialization particles.

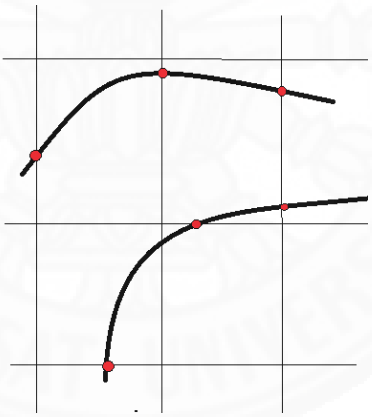


Figure 3.6. Imaginary lines detection initializing points

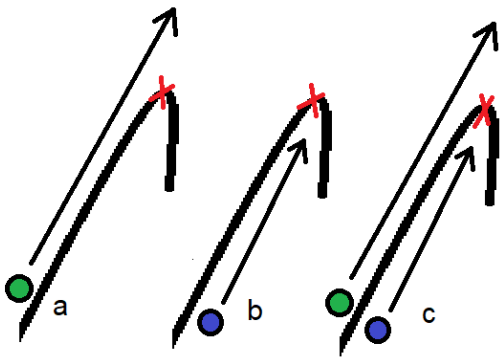


Figure 3.7. Three possible scenarios of initializing particles

Figure 13 (a) shows if only one particle is initialized, then It is not possible to detect the corner due to the reason that particle has never sensed the limit of the free movement since corner is on the other side. If consider the scenario (b) detects the corner since it is on the same side as the initialized particle. Therefore, to predict the correct corner as seen from fig 13 (a) and (b) scenarios is impossible where it needs to initialize two particles side by side to detect the corner. That particular scenario is illustrated in figure 13 (c). This two particle initialization strategy solves the correct corner prediction issue.

3.3.4 Particle movement

The particle is a single pixel programmable object which moves one pixel at a time while sensing its neighboring 8 pixels by using a 9 pixel window that position itself (particle) in the center of the window as shown in the figure 1. Every particle will be assigned a specific direction as an objective. Particles at all times are programmed to achieve its objective and take suitable decisions to achieve this goal. This optimal particle movement and other details about particles are mentioned in section 2.1.

3.3.5 Analyze the data and results

Each and every movement of particles are recorded with any special situations that they faced during its movement, such as turning back, do additional calculations to decide which direction to choose etc. These special situations will give some additional information and helps to detect corners accurately. These records are stored in a SQLite database and algorithm generate conclusions based on analyzing these data. If more than one particle move through same location, it can identify path junctions and these information will be useful for future algorithm expansions, such as detecting junctions or convergence points. Optic disk detection in retinal images is one of the classical examples. Optic disk is a dwelling point that all blood vessels converge at each other. Since particles record each and every movement in the database, query the database to get the highest number of particles in the same location will define the optic disk location.

Chapter 4
Numerical Experiments and the Efficiency

The numerical experiments have been conducted on six synthetic images and algorithm is also tested on a real world problem. Since two separate experiments conducted, this section is divided into two sections. Section 1 presents about synthetic images experiments and section 2 elaborates the real-world application. Section 3 consists technologies used to implement these experiments and hardware configuration of the computer.

4.1 Synthetic image experiments

The first experiment was to show the reliability of the algorithm by using synthetic images. Some of the synthetic images that are used for this experiment is shown in Figures 8-9. All of them are composed of piecewise continuous lines characterized by sharp corners and loops. The accuracy of the corner detection was evaluated for 8 different types of distortions. In these images the ground truth is manually recorded and simply analyses the number of corners detected by HM and STM vs. WPM.

The images were distorted by elliptical lens, horizontal and vertical blinds, curved distortion, bending, ripple, whirl and pinch (Figures 8-9). Additionally a real test image "Block" [5] was used to evaluate the performance of the WPM (Figure 11). The proposed WPM was tested against STM and HM. Since these algorithms require a manual threshold, it was established by the trial and error procedure on the synthetic images without distortion. Initial threshold values are obtained by using images without distortions and keep that threshold level constant to the other test cases. This keeps all algorithms in same threshold-less state to compare with each other.

Then the corner detection is evaluated by the accuracy which is the ratio of the detected corners to the total number of true corners and the false positive ratio (FPR) which is the ratio of the false positives to the total number of true corners [3]. The evaluation results are presented in Table 1.

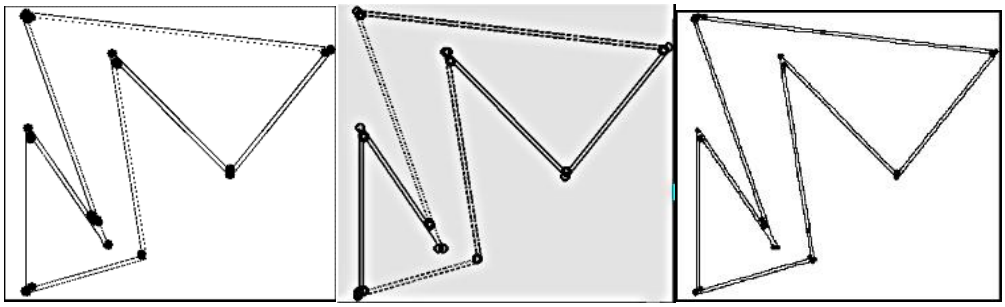


Fig. 4.1. Synthetic images without distortion, WPM, HM, STM

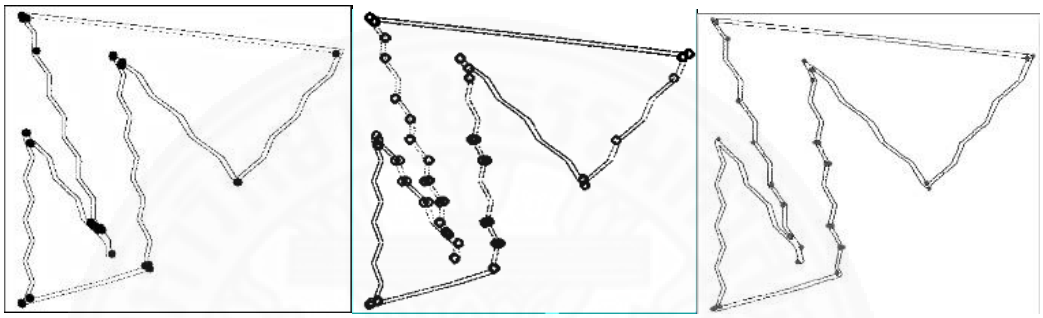


Fig. 4.2. Synthetic images, ripple distortion, WPM, HM, STM

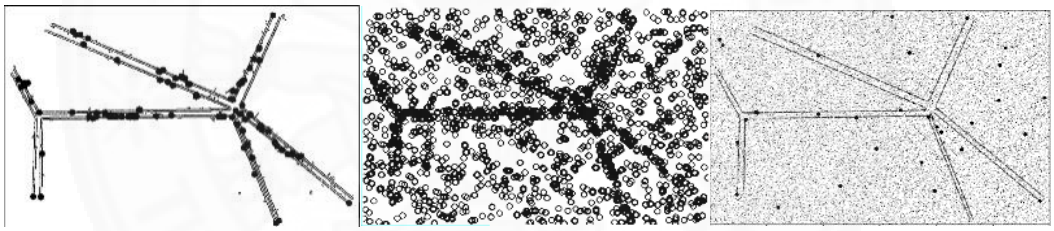


Fig. 4.2. Synthetic noisy images, WPM, HM, STM

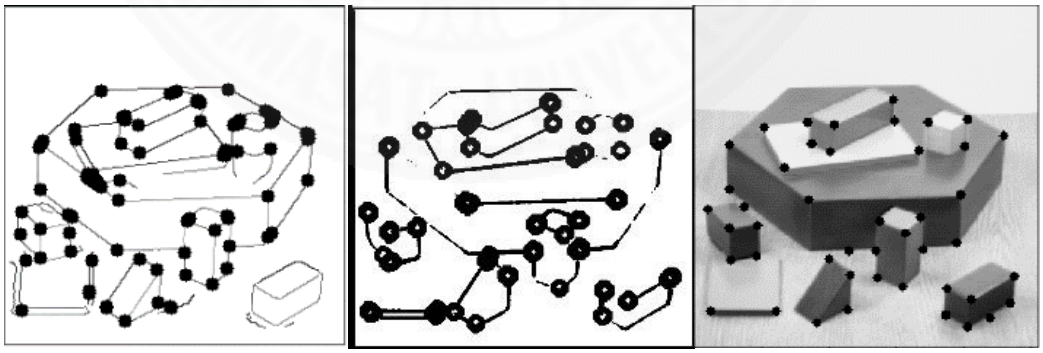


Fig. 4.4. Block image, WPM, HM, STM

Table 4.1. Efficiency of the walking particle method vs. the reference methods

Distortion	Walking particles		Harris		Shi-Tomasi	
	Accuracy,%	False positive, %	Accuracy, %	False positive, %	Accuracy, %	False positive, %
No distortion	94	11	94	6	100	15
Blind 1	81	38	75	40	81	35
Blind 2	81	55	50	66	44	66
Curve	75	47	0	0	83	44
Lens	75	65	63	75	69	52
Ripple	88	17	94	64	88	48
Whirl	81	40	0(failed)	0(failed)	88	41
Noisy (SNR 6.34)	90	21	95	53	90	33
Noisy (SNR 3.54)	55	45	0(failed)	0(failed)	25	80
Average	Average	Average	Average	Average	Average	Average
synthetic	80	35	76	44	75	42
Block image	83	10	61	0	77	6

WP outperform H and ST in 70% of the cases (54 synthetic images and 1 real image). In some cases the WP method does not show the best accuracy, but it detects a significant number of corners with a lesser FP rate. In the ripple distorted image, HM achieve the highest accuracy among other two methods but when compared to the FPR, WPM is the lowest. The STM and WPM record same accuracy but in FPR is significantly lower than the other two methods. 64% and 48% FPR for HM and STM while WPM FPR was only 17%. The HM failed at 33% of the test cases. (To compare with other methods, the failure is registered if the accuracy is less than 5 %). Manual adjustments of the threshold value show better results, but it is not a fair comparison since our method is threshold-less. In 20% of cases HM detects the highest number of true corners, but the FPR is high. The STM shows good results in the distorted images, but fails on the noisy images. For instance, for the SNR 6.34 the STM shows 90% accuracy and 33% FPR while the WPM has the same level of accuracy but 21% FPR. For SNR 3.54 STM shows only 25% accuracy and 80% FPR, whereas WPM beat that with 55% accuracy and 45% FPR. On the average WPM outperform the reference methods for the synthetic images only by 5 and 4% respectively. However, WPM works

much better in the case of the Ripple image and a low SNR (see the Table 1). The WPM also outperform HM and STM on the Block image by 21% and 5% in the accuracy.

4.2 Real world application experiment

The Corner points are also known as feature points and these feature points are coming in handy in many image processing applications [11]. Image registration required to acquire interesting feature points in the source image and the image that need to be registered. For this experiment we used the Matlab software. In Matlab, software FAST [12], Harris [13], SURF [14], KAZE [15], BRISK [16], MSER [17] feature detection algorithms are already available. All these algorithms can be divided into three subcategories based on the feature types. Corner, Blob or region with uniform intensity. Since WPM is a corner detection algorithm, for this experiment only FAST and Harris algorithms are considered because all these feature detection algorithms are based on corners.

The main objective of this experiment is to test WPM in real world applications. In order to perform that, retinal image registration was selected. In Matlab, first need to specify the feature detection algorithm to be used, then can select feature extraction and feature matching algorithms. Since this is about corner detection (feature detection), throughout the experiment the feature extraction and feature matching algorithms are acting similarly. This experiment is not focused on creating a better image registration algorithm, but to find out where our new proposed WPM stands with other corner detection algorithms.

The image registration is carried out by the publicly available FIRE dataset [18]. FIRE dataset consist of 134 image pairs and also divided the images into three classes S, P and A. S class 71 images are having high overlap with no anatomical changes which is the simplest images to register. P class images are having no anatomical change, but smaller overlaps. P class as 49 image pairs. A class is the most difficult to register and high overlaps with high anatomical changes.

WPM implemented using c++ and OpenCV, because of that reason in order to feed to the Matlab feature extraction and feature matching algorithms needs to create a corner point object using the corners detected by the WPM algorithm manually.

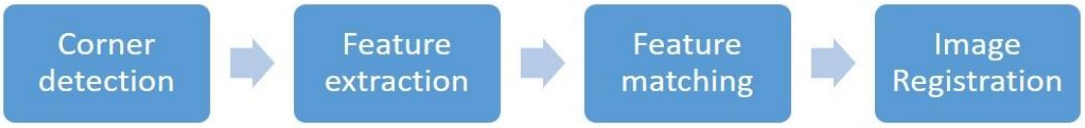
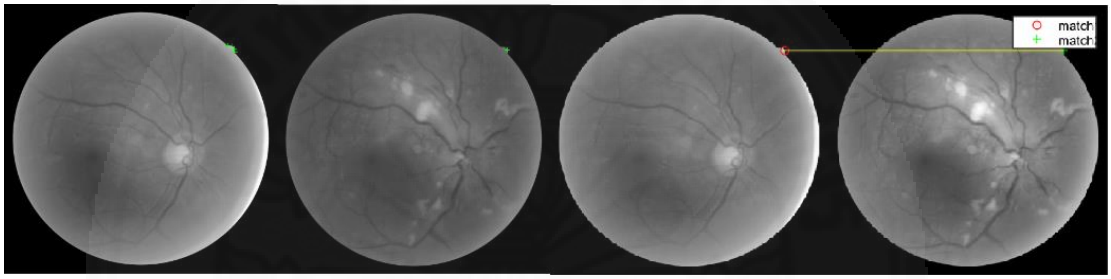
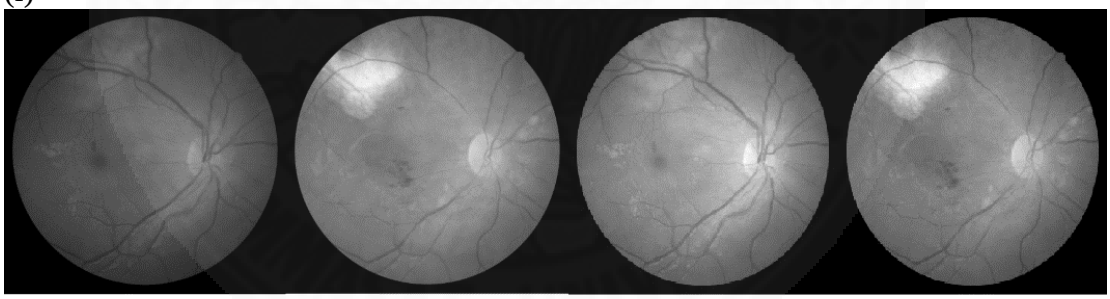


Fig. 4.5. Image registration process

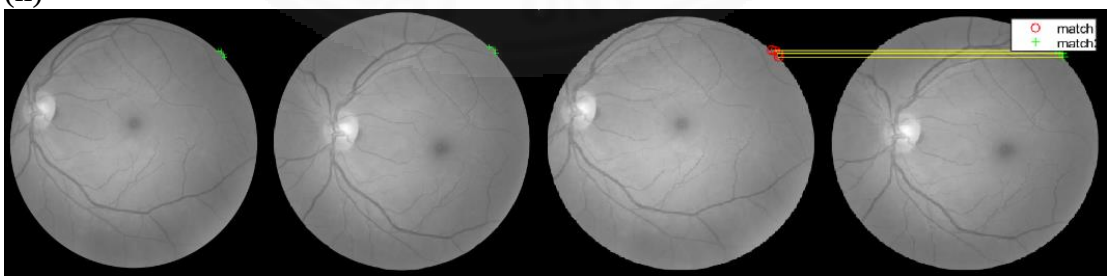
As shown in the Figure 12, Corner detection process is the process only get changed depend on the algorithm that going to be evaluated. WPM corner points are injected into this process through SQLite database. After getting all the corner points, the corner point object is created. This Object then passes to the next process and everything else remained the same for the all other algorithms.



(i)

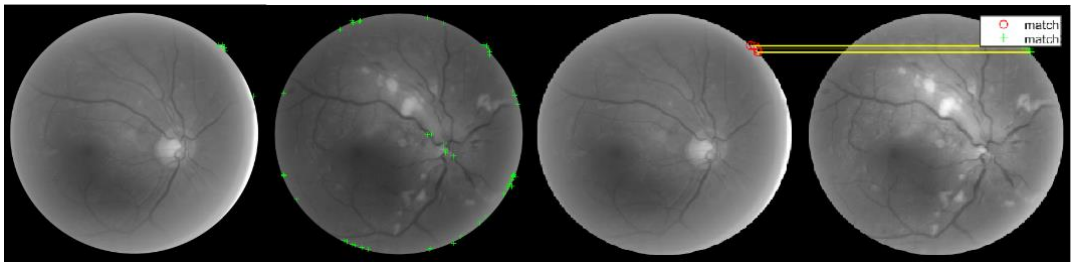


(ii)

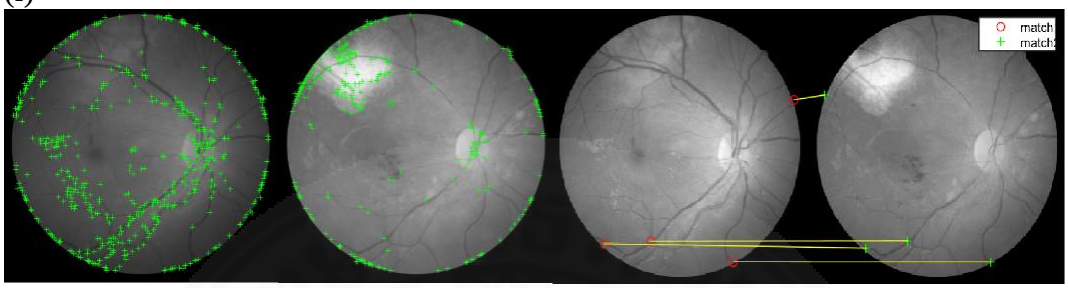


(iii)

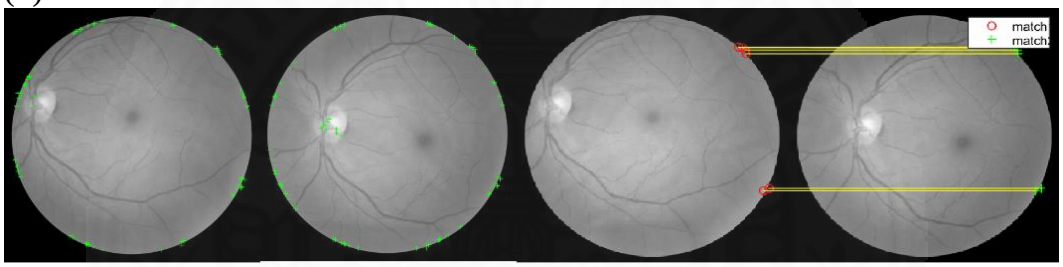
(a) FAST algorithm



(i)

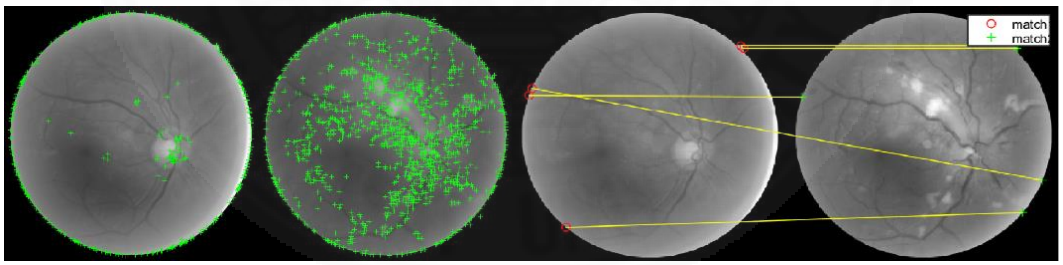


(ii)

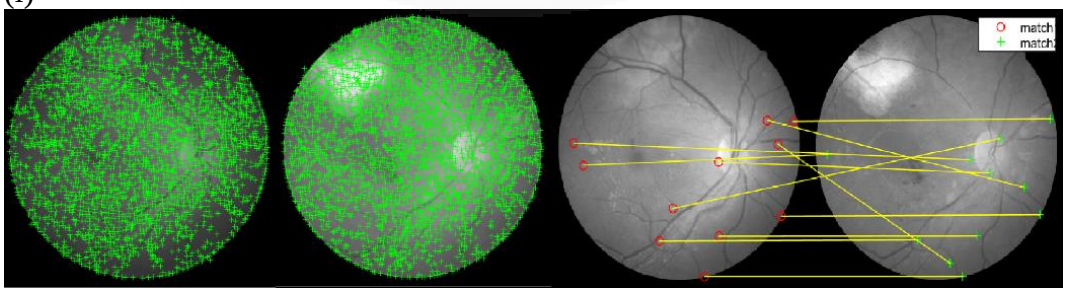


(iii)

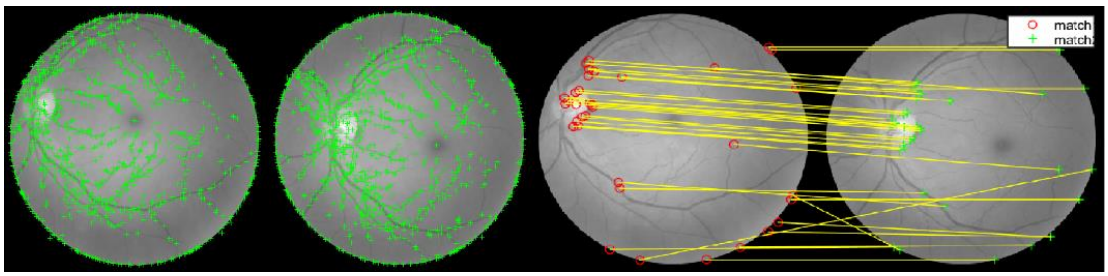
(b) Harris algorithm



(i)



(ii)



(iii)

(c) Proposed method

Fig.4.6 Sample results (first image corner points, second image corner points and matched corners)

In order to evaluate the corner detection method's success, root mean square error (RMSE) is calculated. RMSE error is one of the fundamental accuracy measurement in Image registration [19-21]. Minimum RMSE is the best algorithm. In pursuance to evaluate different algorithms, it first needs to calculate the initial RMSE value for two original images (reference image and image to be registered). After the registration, the registered image and the reference image RMSE value is recalculated. If the new RMSE value is lower than initial original image RMSE value, the registration is classified as a successful registration [19]. Thus in this experiment, corner detection methods are only tested. Due to that reason, successful registrations are very low. In accordance to achieve a successful registration feature detection, extraction and matching processes have to be working seamlessly. This experiment allows to analyze different algorithm's behavior to feature detection in challenging images. Retinal images are categories as challenging images because the intensity and color variations are minimized [20].

Table 4.2. Number of Minimum RMSE

Image Set	WPM	FAST	Harris
A	7	4	3
P	40	1	9
S	44	1	26

Table 4.3. Successful Image Registration

Image Set	WPM	FAST	Harris
A	2	0	0
P	14	1	1
S	11	1	1

Table 4.4. Number of images failed to register.

Image Set	WPM	FAST	Harris
A	0	4	1
P	0	21	3
S	0	29	7

In Table 4.2 shows that WPM outperforms other methods in Minimum RMSE and Table 4.3 represents successful registrations in each image class. Since this experiment does not change feature extraction and matching image registration results are not that successful. Even though WPM outperforms other methods in this comparison as well. Category A, is the most difficult images to register and WPM manages to register 2 images successfully in the 14 images dataset. Compared to Category S, Category P is harder and WPM successfully registers 14 images in that class. WPM also detects at least some corners in every single image while FAST and Harris methods have failed numerously. The main reason behind fail registration is not detecting enough corners so other extraction and matching algorithms cannot function properly. Figure 13, shows some sample images that failed to register and further shows how many corner points are detected by each algorithm. Since HM detects corners based on the intensity difference, most of the time corners are detected around the boundary of the retina. The WPM have good coverage and wide variety of corners because it is initialized many particles and walking around the image. This is the main reason for success of WPM.
After analyzing all the data, it is clear that WPM outperforms Harris and FAST methods in highest number of minimum RMSE, highest number of successful image registration and minimum registration failures.

Chapter 5

Conclusions and future research

The WPM shows promising results, outperforming HM and STM in 70% of the test cases with an average of 80% true corner detection while limits average false corners at 35% in the first experiment. The main advantage of the method is robust for the noisy images and threshold-less corner detection. Second experiment also pointed that when it comes to real world applications proposed WPM can outperform existing algorithms in terms of accuracy. In class (P) images WPM record minimum RMSE in 81% of test cases while more difficult class (A) images more than 50% of the minimum RMSE recorded by WPM. WPM also never failed to register like other compared methods because it always detects significant amount of corners. WPM method proves that it most of the time detect feature rich corners.

The results can be extended to a large image databases as the future research and also can be extended to present complete and more accurate retinal image registration package.

References

1 Rong Wang & Li-qun Gao, 'An Image Registration algorithm based on digital image fusion',16th Triennial World congress In IFAC Publications,Czech Republic,2015,pp. 1018-1021

2 Shyh Wei Teng, Najmus Sadat & Guojun Lu,'Effective and Efficient contour based corner detectors',Pattern Recognition 48,2015,2185-2197

3 Jie Chen, Li-hui Zou, Juan Zhang & Li-hua Dou,'The Comparison and application of corner detection algorithms',Journal of Multimedia,2009,4, (6),pp. 435-441

4 Trupti Patel, Sandip R Panchal,'Corner detection techniques: An introductory survey',International Journal of Engineering Development and Research,2014,2, (4), pp. 3680-3686

5 Wei Chuan Zhang, Peng-Lang Shui,'Contour based corner detection via angle difference of principal directions of anisotropic Gaussian directional derivatives',Pattern Recongnition,2015,48, pp. 2785-2797

6 Bronislav Pribyl, Alan Chalmersmers, Pavel Zemcik,Lucy Hooberman,Martin Cadik,'Evaluation of feature point detection in high dynamic range imagery',Journal of Visual Communication and Image Representation,2016,(38), pp. 141-160

7 Zhijia Zhang, Hongliang Lu, Xin Li,Wenqiang Li & Weiqi Yuan,'Application of improved Harris algorithm in sub pixel feature point extraction',International Journal of Computer and Electrical Engineering ,2014,6,(2), pp. 101-104

8 Adam Schmidt, Marek Kraft and Andrzej Kasinski, 'An evaluation of image feature detectors and descriptors for robot navigation',Proceedings ICCVG 2010,Berlin,2010,Part II pp. 251-259

9 ‘Zero-parameter, automatic canny edge detection with Python and OpenCV’,https://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/, accessed 5 March 2017

10 ‘ORB: an efficient alternative to SIFT or SURF’,http://www.willowgarage.com/sites/default/files/orb_final.pdf,accessed 19 February 2018

11 ‘Local Feature Detection and Extraction’,https://www.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html, accessed 19 February 2018

12 Rosten, E., and T. Drummond,'Machine Learning for High-Speed Corner Detection', 9th European Conference on Computer Vision, 2006, Vol. 1 pp. 430–443

13 Harris, C., and M. J. Stephens,'A Combined Corner and Edge Detector',Proceedings of the 4th Alvey Vision Conference,August 1988,pp. 147–152

14 Leutenegger, S., M. Chli, and R. Siegwart,'BRISK: Binary Robust Invariant Scalable Keypoints', Proceedings of the IEEE International Conference. ICCV,Barcelona, Spain,Nov. 2011,pp. 2548-2555

15 Matas, J., O. Chum, M. Urba, and T. Pajdla, 'Robust wide-baseline stereo from maximally stable extremal regions',Proceedings of British Machine Vision Conference,UK,2002,pp. 384–396

16 Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool,'SURF: Speeded Up Robust Features',Computer Vision and Image Understanding (CVIU),2008,110,3,pp. 346–359

17 Alcantarilla, P.F., A. Bartoli, and A.J. Davison,'KAZE Features',ECCV 2012,2012,VI,7577,pp. 214

18 C. Hernandez-Matas, X. Zabulis, A. Triantafyllou, P. Anyfanti, S. Douma, A.A. Argyros,'Journal for Modeling in Ophthalmology',2017,1,4, pp. 16-28

19 Ghassabi, Z., Shanbehzadeh, J., Sedaghat, A. et al.,'An efficient approach for robust multimodal retinal image registration based on UR-SIFT features and PIIFD descriptors',Journal on Image and Video Processing,2013, 2013,25, pp. 1-16

20 GK Matsopoulos, MPA Asvestas, NA Mouravliansky, KK Delibasis,'Multimodal registration of retinal images using self-organizing maps',IEEE Trans. on Med. Imaging,2004,12,pp. 1557–1563

21 L Jupeng, C Houjin, Y Chang, Z Xinyuan, 'robust feature-based method for mosaic of the curved human color retinal images', Proceedings of Biomedical Engineering and Informatics,Sanya, Hainan,May 2008,pp. 27–30

22 Yi-bo, Li & Jun-jun, Li,'Harris Corner Detection Algorithm Based on Improved Contourlet Transform',Procedia Engineering,2011,08,419,pp. 2239-2243

Appendices

Appendix A
List of Publication

Pasindu Kuruppuarachchi and Stanislav S. Makhanov. (2018). Corner Detection via Walking Particles, ICCET '18 Proceedings of the 2018 International Conference on Communication Engineering and Technology Singapore. 2018, pp. 15-17. https://doi.org/10.1145/3194244.3194252

Appendix B

Algorithm code

```
#include "stdafx.h"
#include "main.h"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
#include <iterator>
#include <thread>
#include <vector>
#include <sqlite\sqlite3.h>
#include <time.h>

#define w 400

using namespace cv;
using namespace std;


#pragma region Global Variables

Mat src, src_gray;
Mat dst, detected_edges;
Mat new_image; // color enhaced image
Mat read_image;//image that read from the file
Mat intersection_image;

int edgeThresh = 1;
int lowThreshold = 12;//for test
int const max_lowThreshold = 200;
int ratios = 3;
int kernel_size = 3;
char* window_name = "Edge Map";
char* window_name2 = "Color Enhancement";

double alpha = 1; /**< Simple contrast control alpha value [1.0-3.0]*/
int beta = 0;  /**< Simple brightness control beta value [0-100]*/

int imageRows = 0;
int imageColumns = 0;

ofstream edgeMapFile;
ofstream particalPathfile;

vector<vector<int> > v2d;
vector<int> tempVector; //col
vector<vector<int> > edgeMapVector; //rows

vector<vector<int> > leftSeedPoints;
```

```
vector<vector<int> > rightSeedPoints;
vector<vector<int> > upSeedPoints;
vector<vector<int> > downSeedpoint;

vector<int> pathCoordinatesX;
vector<int> pathCoordinatesY;


Mat image = imread("iphone images/IMG_0017.jpg");

sqlite3 *db;
char *zErrMsg = 0;
int rc;
char *sql;

int similarMovementRetValue = 0;
char direction;


vector<Point2f> pts_src;
vector<Point2f> pts_dst;

#pragma endregion


#pragma region Methods()

void insertCornerPoints(int particalID, int x, int y);
void CheckCornerPoints(int particalID, int x, int y,int blockCategory, char direction);
int checkEmptySpace(int i, int j, char direction);

/*Possible Return types U UL UR D DL DR L R */
/*Only four possible movement types L R U D*/

void CheckCornerPoints(int particalID, int x, int y, int blockCategory,char direction)
{
	/* blockCategory
		=1	three consecutive blocks
		=2	four blocks and looking for less than 10 count for all four directions
		=3	three blocks and less than 1 to 2 counts atleast 3 directions
		=4	five blocks or more filled*/
	int l = 0;
	int r = 0;
	int u = 0;
	int d = 0;
	int directionCount = 0;
	int case1Limit = 3;//5
	int case2Limit = 3;//4
	int case3Limit = 2;//2
	int case4Limit = 3;

	int case2Limit2 = 0;//4
	int case3Limit2 = 0;//2
	int case4Limit2 = 0;

	int left = 0;
	int right = 0;
```

```
int up = 0;
int down = 0;


switch (blockCategory)
{
case 1:
	switch (direction)
	{
	case 'U':
		l = checkEmptySpace(x, y, 'L');
		r = checkEmptySpace(x, y, 'R');
		if (l <= case1Limit && r <= case1Limit)
		{
			insertCornerPoints(particalID, x, y);
		}
		else
		{
			CheckCornerPoints(particalID, x, y, 3, direction);
		}
		break;
	case 'D':
		l = checkEmptySpace(x, y, 'L');
		r = checkEmptySpace(x, y, 'R');
		if (l <= case1Limit && r <= case1Limit)
		{
			insertCornerPoints(particalID, x, y);
		}
		else
		{
			CheckCornerPoints(particalID, x, y, 3, direction);
		}
		break;
	case 'R':
		u = checkEmptySpace(x, y, 'U');
		d = checkEmptySpace(x, y, 'D');
		if (u <= case1Limit && d <= case1Limit)
		{
			insertCornerPoints(particalID, x, y);
		}
		else
		{
			CheckCornerPoints(particalID, x, y, 3, direction);
		}
		break;
	case 'L':
		u = checkEmptySpace(x, y, 'U');
		d = checkEmptySpace(x, y, 'D');
		if (u <= case1Limit && d <= case1Limit)
		{
			insertCornerPoints(particalID, x, y);
		}
		else
		{
			CheckCornerPoints(particalID, x, y, 3, direction);
		}
		break;
```

```
		default:
			break;
		}
		break;
	case 2:
		l = checkEmptySpace(x, y, 'L');
		r = checkEmptySpace(x, y, 'R');
		d = checkEmptySpace(x, y, 'D');
		u = checkEmptySpace(x, y, 'U');
		/*if(l<=10 && r <= 10 && d <= 10 && u <= 10)
			insertCornerPoints(particalID, x, y);*/
		if (l <= case2Limit)
		{
			directionCount++;
			left = 1;
		}


		if (r <= case2Limit)
		{
			directionCount++;
			right = 1;
		}

		if (d <= case2Limit)
		{
			directionCount++;
			down = 1;
		}

		if (u <= case2Limit)
		{
			directionCount++;
			up = 1;
		}

		if (directionCount >= 3)
		{
			if(directionCount != 4 && left == 1 && right == 1 && up == 1 && down
<= case2Limit2)
				insertCornerPoints(particalID, x, y);
			else if (directionCount != 4 && left == 1 && right == 1 && down == 1 &&
up <= case2Limit2)
				insertCornerPoints(particalID, x, y);
			else if (directionCount != 4 && left == 1 &&  down == 1 && up == 1 &&
right <= case2Limit2)
				insertCornerPoints(particalID, x, y);
			else if (directionCount != 4 &&  down == 1 && right == 1 && up == 1 &&
left <= case2Limit2)
				insertCornerPoints(particalID, x, y);
			else
				insertCornerPoints(particalID, x, y);
		}
		break;
	case 3:
		l = checkEmptySpace(x, y, 'L');
		r = checkEmptySpace(x, y, 'R');
```

```
                d = checkEmptySpace(x, y, 'D');
                u = checkEmptySpace(x, y, 'U');

                if (l <= case3Limit)
                        directionCount++;

                if (r <= case3Limit)
                        directionCount++;

                if (d <= case3Limit)
                        directionCount++;

                if (u <= case3Limit)
                        directionCount++;

                if (directionCount >= 3)
                {
                        if (directionCount != 4 && left == 1 && right == 1 && up == 1 && down
<= case3Limit2)
                                insertCornerPoints(particalID, x, y);
                        else if (directionCount != 4 && left == 1 && right == 1 && down == 1 &&
up <= case3Limit2)
                                insertCornerPoints(particalID, x, y);
                        else if (directionCount != 4 && left == 1 && down == 1 && up == 1 &&
right <= case3Limit2)
                                insertCornerPoints(particalID, x, y);
                        else if (directionCount != 4 && down == 1 && right == 1 && up == 1 &&
left <= case3Limit2)
                                insertCornerPoints(particalID, x, y);
                        else
                                insertCornerPoints(particalID, x, y);
                }
                break;
        case 4:
                l = checkEmptySpace(x, y, 'L');
                r = checkEmptySpace(x, y, 'R');
                d = checkEmptySpace(x, y, 'D');
                u = checkEmptySpace(x, y, 'U');
                /*if(l<=10 && r <= 10 && d <= 10 && u <= 10)
                insertCornerPoints(particalID, x, y);*/
                if (l <= case4Limit)
                        directionCount++;

                if (r <= case4Limit)
                        directionCount++;

                if (d <= case4Limit)
                        directionCount++;

                if (u <= case4Limit)
                        directionCount++;

                if (directionCount >= 3)
                {
                        if (directionCount != 4 && left == 1 && right == 1 && up == 1 && down
<= case4Limit2)
                                insertCornerPoints(particalID, x, y);
```

```
                        else if (directionCount != 4 && left == 1 && right == 1 && down == 1 &&
up <= case4Limit2)
                                insertCornerPoints(particalID, x, y);
                        else if (directionCount != 4 && left == 1 && down == 1 && up == 1 &&
right <= case4Limit2)
                                insertCornerPoints(particalID, x, y);
                        else if (directionCount != 4 && down == 1 && right == 1 && up == 1 &&
left <= case4Limit2)
                                insertCornerPoints(particalID, x, y);
                        else
                                insertCornerPoints(particalID, x, y);
                }
                break;
        default:
                break;
        }
}

int checkEmptySpace(int i, int j, char direction)
{
        cout << "check empty " << to_string(i) << "\t" << to_string(j) << endl;
        int temp = 0;
        int count = 1;
        switch (direction) {
        case 'U':
                while (temp != 255)
                {
                        if (i - count >= 0)
                        {
                                //cout << "check empty " << to_string(i) << "\t" << to_string(j) <<
endl;
                                temp = edgeMapVector[i - count][j];
                                count++;
                        }
                        else
                        {
                                break;
                        }

                }
                break;
        case 'D':
                while (temp != 255)
                {
                        if (i + count < edgeMapVector.size())
                        {
                                //cout << "check empty " << to_string(i) << "\t" << to_string(j) <<
endl;
                                temp = edgeMapVector[i + count][j];
                                count++;
                        }
                        else
                        {
                                break;
                        }

                }
```

```
			break;
		case 'L':
			while (temp != 255)
			{
				if (j - count >= 0)
				{
					//cout << "check empty " << to_string(i) << "\t" << to_string(j) <<
endl;
					temp = edgeMapVector[i][j - count];
					count++;
				}
				else
				{
					break;
				}

			}
			break;
		case 'R':
			while (temp != 255)
			{
				if (j + count < edgeMapVector[0].size())
				{
					//cout << "check empty " << to_string(i) << "\t" << to_string(j) <<
endl;
					temp = edgeMapVector[i][j + count];
					count++;
				}
				else
				{
					break;
				}
			}
			break;
		}
		cout << "check empty " << direction << " ret" << endl;
		return count-2; //in the loop we have count++ so then loop exit count is +1 and also we start
from count=1 to remove that need to add -2 in the last return

}

char* isValidMoveAvailable(int pid, int i, int j, char particalDirection)
{

#pragma region CheckCorner Logic without movement condition

		int countOfBlocks=0;

		if (i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j - 1
>= 0 && edgeMapVector[i - 1][j - 1] == 255)
		{
			countOfBlocks++;
		}

		if (i < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i >= 0 && j - 1 >= 0
&& edgeMapVector[i][j - 1] == 255)
		{
```

```
		countOfBlocks++;
	}

	if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && j - 1
>= 0 && edgeMapVector[i + 1][j - 1] == 255)
	{
		countOfBlocks++;
	}

	if (i - 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i - 1 >= 0 && j >= 0
&& edgeMapVector[i - 1][j] == 255)
	{
		countOfBlocks++;
	}

	if (i + 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i + 1 >= 0 && j >= 0
&& edgeMapVector[i + 1][j] == 255)
	{
		countOfBlocks++;
	}

	if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1
>= 0 && edgeMapVector[i - 1][j + 1] == 255)
	{
		countOfBlocks++;
	}

	if (i < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i >= 0 && j + 1 >= 0
&& edgeMapVector[i][j + 1] == 255)
	{
		countOfBlocks++;
	}

	if (i + 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && j +
1 >= 0 && edgeMapVector[i + 1][j + 1] == 255)
	{
		countOfBlocks++;
	}

	if (countOfBlocks == 4)
	{
		switch (direction)
		{
			case 'U':
			if (i - 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i - 1
>= 0 && j >= 0 && edgeMapVector[i - 1][j] == 255)
				CheckCornerPoints(pid, i, j, 2, direction);
			break;
			case 'D':
				if (i + 1 < edgeMapVector.size() && j < edgeMapVector[0].size()
&& i + 1 >= 0 && j >= 0 && edgeMapVector[i + 1][j] == 255)
					CheckCornerPoints(pid, i, j, 2, direction);
				break;
			case 'R':
				if (i < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& i >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255)
					CheckCornerPoints(pid, i, j, 2, direction);
```

```
                                        break;
                                case 'L':
                                        if (i < edgeMapVector.size() && j - 1 < edgeMapVector[0].size()
&& i >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255)
                                                CheckCornerPoints(pid, i, j, 2, direction);
                                        break;
                        }

                }
                else if (countOfBlocks >= 5)
                {
                        switch (direction)
                        {
                        case 'U':
                                if (i - 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i - 1
>= 0 && j >= 0 && edgeMapVector[i - 1][j] == 255)
                                        CheckCornerPoints(pid, i, j, 4, direction);
                                break;
                        case 'D':
                                if (i + 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i + 1
>= 0 && j >= 0 && edgeMapVector[i + 1][j] == 255)
                                        CheckCornerPoints(pid, i, j, 4, direction);
                                break;
                        case 'R':
                                if (i < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i >=
0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255)
                                        CheckCornerPoints(pid, i, j, 4, direction);
                                break;
                        case 'L':
                                if (i < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i >= 0
&& j - 1 >= 0 && edgeMapVector[i][j - 1] == 255)
                                        CheckCornerPoints(pid, i, j, 4, direction);
                                break;
                        }

                }
                else if (countOfBlocks == 3)
                {
                        switch (direction)
                        {
                        case 'U':
                                if (i - 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i - 1
>= 0 && j >= 0 && edgeMapVector[i - 1][j] == 255)
                                        CheckCornerPoints(pid, i, j, 3, direction);
                                break;
                        case 'D':
                                if (i + 1 < edgeMapVector.size() && j < edgeMapVector[0].size() && i + 1
>= 0 && j >= 0 && edgeMapVector[i + 1][j] == 255)
                                        CheckCornerPoints(pid, i, j, 3, direction);
                                break;
                        case 'R':
                                if (i < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i >=
0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255)
                                        CheckCornerPoints(pid, i, j, 3, direction);
                                break;
                        case 'L':
```

```
                        if (i < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i >= 0
&& j - 1 >= 0 && edgeMapVector[i][j - 1] == 255)
                                CheckCornerPoints(pid, i, j, 3, direction);
                        break;
                }
        }
#pragma endregion

        switch (direction) {
        case 'U':
                if (i - 1 <= 0 || j + 1 >= edgeMapVector[0].size() || j - 1 <= 0)
                {
                        return "0";
                }
                else
                {

                        if (j < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j] != 255)
                        {
                                //cout << to_string(i) << "\t" << to_string(j) << "\tU_4" << endl;
                                return "U";
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j - 1] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j] == 255 &&
edgeMapVector[i][j + 1] != 255 && edgeMapVector[i][j - 1] != 255)
                        {
                                CheckCornerPoints(pid, i, j, 1, 'U');
                                int c1 = checkEmptySpace(i, j, 'L');
                                int c2 = checkEmptySpace(i, j, 'R');

                                if (c1 >= c2)
                                {
                                        direction = 'L';
                                        return "L";
                                }
                                else
                                {
                                        direction = 'R';
                                        return "R";
                                }

                        }//5
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 &&
edgeMapVector[i][j - 1] == 255 && edgeMapVector[i][j + 1] != 255)
                        {
                                direction = 'R';
                                CheckCornerPoints(pid, i, j, 2, 'U');
                                /*insertCornerPoints(pid, i, j + 1);*/
                                //cout << to_string(i) << "\t" << to_string(j) << "\tR_41" << endl;
                                return "R";//5
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
```

```
- 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 &&
edgeMapVector[i][j + 1] == 255 && edgeMapVector[i][j - 1] != 255)
                        {
                                direction = 'L';
                                CheckCornerPoints(pid, i, j, 2, 'U');
                                /*insertCornerPoints(pid, i, j - 1);*/
                                //cout << to_string(i) << "\t" << to_string(j) << "\tL_42" << endl;
                                return "L";//5
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i][j + 1] == 255 &&
edgeMapVector[i - 1][j - 1] != 255)
                        {
                                CheckCornerPoints(pid, i, j, 3, 'U');
                                //cout << to_string(i) << "\t" << to_string(j) << "\tUL_43" << endl;
                                return "UL";//4
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i][j - 1] == 255 &&
edgeMapVector[i - 1][j + 1] != 255)
                        {
                                CheckCornerPoints(pid, i, j, 3, 'U');
                                //cout << to_string(i) << "\t" << to_string(j) << "\tUR_44" << endl;
                                return "UR";//4
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i][j - 1] == 255 &&
edgeMapVector[i][j + 1] != 255)
                        {
                                direction = 'R';
                                CheckCornerPoints(pid, i, j, 3, 'U');
                                /*insertCornerPoints(pid, i, j + 1);*/
                                //cout << to_string(i) << "\t" << to_string(j) << "\tR_45" << endl;
                                return "R";//4
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i][j + 1] == 255 &&
edgeMapVector[i][j - 1] != 255)
                        {
                                direction = 'L';
                                CheckCornerPoints(pid, i, j, 3, 'U');
                                /*insertCornerPoints(pid, i, j - 1);*/
                                //cout << to_string(i) << "\t" << to_string(j) << "\tL_46" << endl;
                                return "L";//4
                        }
                        else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i][j - 1] == 255)
                        {
                                direction = 'D';
                                CheckCornerPoints(pid, i, j, 3, 'U');
                                /*insertCornerPoints(pid, i + 1, j);*/
                                //cout << to_string(i) << "\t" << to_string(j) << "\tD_47" << endl;
```

```
                return "D";//3
            }
            else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j + 1] != 255)
            {

                //cout << to_string(i) << "\t" << to_string(j) << "\tUR_48" << endl;
                return "UR";//3
            }
            else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i - 1][j - 1] != 255)
            {
                //cout << to_string(i) << "\t" << to_string(j) << "\tUL_49" << endl;
                return "UL";//3
            }
            else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j - 1] != 255)
            {
                //cout << to_string(i) << "\t" << to_string(j) << "\tUL_491" <<
endl;
                return "UL";//3
            }
            else if (i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size()
&& j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i
- 1][j] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i - 1][j + 1] != 255)
            {
                //cout << to_string(i) << "\t" << to_string(j) << "\tUR_492" <<
endl;
                return "UR";//3
            }
            else if (j + 1 < edgeMapVector[0].size() && i + 1 < edgeMapVector.size()
&& edgeMapVector[i + 1][j + 1] != 255)
            {
                direction = 'D';
                //cout << to_string(i) << "\t" << to_string(j) << "\tDR_493" <<
endl;
                return "DR";
            }
            else if (j - 1 < edgeMapVector[0].size() && i + 1 < edgeMapVector.size()
&& edgeMapVector[i + 1][j - 1] != 255)
            {
                direction = 'D';
                //cout << to_string(i) << "\t" << to_string(j) << "\tDL_494" <<
endl;
                return "DL";
            }
            else if (j < edgeMapVector[0].size() && i + 1 < edgeMapVector.size() &&
edgeMapVector[i + 1][j] != 255)
            {
                direction = 'D';
                /*insertCornerPoints(pid, i + 1, j);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tD_495" << endl;
                return "D";
            }
```

```
                else
                        return "0";
        }
        break;
case 'D':
        if (i + 1 >= edgeMapVector.size() || j + 1 >= edgeMapVector[0].size() || j - 1 <= 0)
        {
                return "0";
        }
        else if (j < edgeMapVector[0].size() && i + 1 < edgeMapVector.size() &&
edgeMapVector[i + 1][j] != 255)
        {
                //cout << to_string(i) << "\t" << to_string(j) << "\tD_3" << endl;
                return "D";
        }
        else if (i + 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && j - 1
< edgeMapVector[0].size() && i + 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i + 1][j -
1] == 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i + 1][j] == 255 &&
edgeMapVector[i][j + 1] != 255 && edgeMapVector[i][j - 1] != 255)
        {
                CheckCornerPoints(pid, i, j, 1, 'D');

                int c1 = checkEmptySpace(i, j, 'L');
                int c2 = checkEmptySpace(i, j, 'R');

                if (c1 >= c2)
                {
                        direction = 'L';
                        return "L";
                }
                else
                {
                        direction = 'R';
                        return "R";
                }

        }//5
        else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i + 1][j + 1] == 255 &&
edgeMapVector[i][j + 1] == 255 && edgeMapVector[i][j - 1] != 255)
        {
                direction = 'L';
                CheckCornerPoints(pid, i, j, 2, 'D');
                /*insertCornerPoints(pid, i, j - 1);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tL_31" << endl;
                return "L";//4
        }
        else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i + 1][j + 1] == 255 &&
edgeMapVector[i][j - 1] == 255 && edgeMapVector[i][j + 1] != 255)
        {
                direction = 'R';
                CheckCornerPoints(pid, i, j, 2, 'D');
                /*insertCornerPoints(pid, i, j + 1);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tR_32" << endl;
```

```
                return "R";//4
            }
            else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i][j - 1] == 255 &&
edgeMapVector[i + 1][j + 1] != 255)
            {
                CheckCornerPoints(pid, i, j, 3, 'D');
                //cout << to_string(i) << "\t" << to_string(j) << "\tDR_33" << endl;
                return "DR";//4
            }
            else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i][j + 1] == 255 &&
edgeMapVector[i + 1][j - 1] != 255)
            {
                CheckCornerPoints(pid, i, j, 3, 'D');
                //cout << to_string(i) << "\t" << to_string(j) << "\tDL_34" << endl;
                return "DL";//4
            }
            else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i][j - 1] == 255 &&
edgeMapVector[i][j + 1] != 255)
            {
                direction = 'R';
                CheckCornerPoints(pid, i, j, 3, 'D');
                /*insertCornerPoints(pid, i, j + 1);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tR_35" << endl;
                return "R";//4
            }
            else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i][j + 1] == 255 &&
edgeMapVector[i][j - 1] != 255)
            {
                direction = 'L';
                CheckCornerPoints(pid, i, j, 3, 'D');
                /*insertCornerPoints(pid, i, j - 1);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tL_36" << endl;
                return "L";//4
            }
            else if (i + 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && j - 1
< edgeMapVector[0].size() && i + 1 >= 0 && j + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i][j - 1] == 255)
            {
                direction = 'U';
                CheckCornerPoints(pid, i, j, 3, 'D');
                /*insertCornerPoints(pid, i - 1, j);*/
                //cout << to_string(i) << "\t" << to_string(j) << "\tU_37" << endl;
                return "U";//3
            }
            else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i + 1][j + 1] != 255)
            {
                //cout << to_string(i) << "\t" << to_string(j) << "\tDR_38" << endl;
```

```
                        return "DR";//3
                }
                else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i + 1][j - 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDL_39" << endl;
                        return "DL";//3
                }
                else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i + 1][j + 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDR_391" << endl;
                        return "DR";//3
                }
                else if (i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && j + 1
< edgeMapVector[0].size() && i + 1 >= 0 && j - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i + 1][j]
== 255 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i + 1][j - 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDL_392" << endl;
                        return "DL";//3
                }
                else if (j + 1 < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j + 1] != 255)
                {
                        direction = 'U';
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUR_393" << endl;
                        return "UR";
                }
                else if (j - 1 < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j - 1] != 255)
                {
                        direction = 'U';
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUL_394" << endl;
                        return "UL";
                }
                else if (j < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j] != 255)
                {
                        direction = 'U';
                        /*insertCornerPoints(pid, i + 1, j);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tU_395" << endl;
                        return "U";
                }
                else
                        return "0";
                break;
        case 'L':
                if (j - 1 <= 0 || i + 1 >= edgeMapVector.size() || i - 1 <= 0)
                {
                        return "0";
                }
                else if (j - 1 < edgeMapVector[0].size() && i < edgeMapVector.size() &&
edgeMapVector[i][j - 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tL_2" << endl;
```

```
			return "L";
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i + 1][j] != 255 && edgeMapVector[i - 1][j] != 255)
		{
			CheckCornerPoints(pid, i, j, 1, 'L');
			int c1 = checkEmptySpace(i, j, 'U');
			int c2 = checkEmptySpace(i, j, 'D');

			if (c1 >= c2)
			{
				direction = 'U';
				return "U";
			}
			else
			{
				direction = 'D';
				return "D";
			}

		}//5
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i + 1][j - 1] && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j] != 255)
		{
			direction = 'U';
			CheckCornerPoints(pid, i, j, 2, 'L');
			/*insertCornerPoints(pid, i - 1, j);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tU_21" << endl;
			return "U";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i + 1][j - 1] && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j] != 255)
		{
			direction = 'D';
			CheckCornerPoints(pid, i, j, 2, 'L');
			/*insertCornerPoints(pid, i + 1, j);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tD_22" << endl;
			return "D";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j] != 255)
		{
			direction = 'D';
			CheckCornerPoints(pid, i, j, 3, 'L');
			/*insertCornerPoints(pid, i + 1, j);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tD_23" << endl;
			return "D";//4
		}
```

```
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j] != 255)
		{
			direction = 'U';
			CheckCornerPoints(pid, i, j, 3, 'L');
			/*insertCornerPoints(pid, i - 1, j);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tU_24" << endl;
			return "U";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j - 1] != 255)
		{
			CheckCornerPoints(pid, i, j, 3, 'L');
			//cout << to_string(i) << "\t" << to_string(j) << "\tUL_25" << endl;
			return "UL";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j - 1] != 255)
		{
			CheckCornerPoints(pid, i, j, 3, 'L');
			//cout << to_string(i) << "\t" << to_string(j) << "\tDL_26" << endl;
			return "DL";//4
		}
		else if (i - 1 < edgeMapVector.size() && i + 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i - 1 >= 0 && i + 1 >= 0 && j - 1 >= 0 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i][j - 1] == 255)
		{
			CheckCornerPoints(pid, i, j, 3, 'L');
			direction = 'R';
			/*insertCornerPoints(pid, i, j + 1);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tR_27" << endl;
			return "R";//3
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j - 1] != 255)
		{
			//cout << to_string(i) << "\t" << to_string(j) << "\tDL_28" << endl;
			return "DL";//3
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j - 1] != 255)
		{
			//cout << to_string(i) << "\t" << to_string(j) << "\tUL_29" << endl;
			return "UL";//3
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] == 255 && edgeMapVector[i + 1][j - 1] == 255 && edgeMapVector[i - 1][j - 1] != 255)
		{
```

```
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUL_291" << endl;
                        return "UL";//3
                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j - 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j - 1 >= 0 && edgeMapVector[i][j - 1] ==
255 && edgeMapVector[i - 1][j - 1] == 255 && edgeMapVector[i + 1][j - 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDL_292" << endl;
                        return "DL";//3
                }
                else if (j + 1 < edgeMapVector[0].size() && i + 1 < edgeMapVector.size() &&
edgeMapVector[i + 1][j + 1] != 255)
                {
                        direction = 'R';
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDR_293" << endl;
                        return "DR";
                }
                else if (j + 1 < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j + 1] != 255)
                {
                        direction = 'R';
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUR_294" << endl;
                        return "UR";
                }
                else if (j + 1 < edgeMapVector[0].size() && i < edgeMapVector.size() &&
edgeMapVector[i][j + 1] != 255)
                {
                        direction = 'R';
                        /*insertCornerPoints(pid, i, j + 1);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tR_295" << endl;
                        return "R";
                }
                else
                        return "0";
                break;
        case 'R':
                if (j + 1 >= edgeMapVector[0].size() || i + 1 >= edgeMapVector.size() || i - 1 <= 0)
                {
                        return "0";
                }
                if (j + 1 < edgeMapVector[0].size() && i < edgeMapVector.size() &&
edgeMapVector[i][j + 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tR_1" << endl;
                        return "R"; //0
                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] ==
255 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 &&
edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i + 1][j] != 255)
                {
                        direction = 'D';
                        CheckCornerPoints(pid, i, j, 2, 'R');
                        /*insertCornerPoints(pid, i + 1, j);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tD_11" << endl;
                        return "D";//5
                }
```

```
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j] != 255)
		{
			direction = 'U';
			CheckCornerPoints(pid, i, j, 2, 'R');
			/*insertCornerPoints(pid, i - 1, j);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tU_12" << endl;
			return "U";//5
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i + 1][j] != 255 && edgeMapVector[i - 1][j] != 255)
		{
			CheckCornerPoints(pid, i, j, 1, 'R');
			int c1 = checkEmptySpace(i, j, 'U');
			int c2 = checkEmptySpace(i, j, 'D');

			if (c1 >= c2)
			{
				direction = 'U';
				return "U";
			}
			else
			{
				direction = 'D';
				return "D";
			}

		}//5
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j + 1] != 255)
		{
			CheckCornerPoints(pid, i, j, 3, 'R');
			//cout << to_string(i) << "\t" << to_string(j) << "\tUR_13" << endl;
			return "UR";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i - 1][j] == 255 && edgeMapVector[i + 1][j + 1] != 255)
		{
			CheckCornerPoints(pid, i, j, 3, 'R');
			//cout << to_string(i) << "\t" << to_string(j) << "\tDR_14" << endl;
			return "DR";//4
		}
		else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 < edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] == 255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j] != 255)
		{
			direction = 'U';
			CheckCornerPoints(pid, i, j, 3, 'R');
```

```
                        /*insertCornerPoints(pid, i - 1, j);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tU_15" << endl;
                        return "U";//4
                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] ==
255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i - 1][j + 1] == 255 &&
edgeMapVector[i - 1][j] != 255)
                {
                        direction = 'U';
                        CheckCornerPoints(pid, i, j, 3, 'R');
                        /*insertCornerPoints(pid, i - 1, j);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tU_16" << endl;
                        return "U";//4
                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] ==
255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i - 1][j] == 255 &&
edgeMapVector[i + 1][j] != 255)
                {
                        direction = 'D';
                        CheckCornerPoints(pid, i, j, 3, 'R');
                        /*insertCornerPoints(pid, i + 1, j);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tD_17" << endl;
                        return "D";//4
                }
                else if (i - 1 < edgeMapVector.size() && i + 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i - 1 >= 0 && i + 1 >= 0 && j + 1 >= 0 && edgeMapVector[i - 1][j] ==
255 && edgeMapVector[i + 1][j] == 255 && edgeMapVector[i][j + 1] == 255)
                {
                        direction = 'L';
                        CheckCornerPoints(pid, i, j, 3, 'R');
                        /*insertCornerPoints(pid, i, j - 1);*/
                        //cout << to_string(i) << "\t" << to_string(j) << "\tL_18" << endl;
                        return "L";//3

                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] ==
255 && edgeMapVector[i + 1][j + 1] == 255 && edgeMapVector[i - 1][j + 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUR_19" << endl;
                        return "UR";//3
                }
                else if (i + 1 < edgeMapVector.size() && i - 1 < edgeMapVector.size() && j + 1 <
edgeMapVector[0].size() && i + 1 >= 0 && i - 1 >= 0 && j + 1 >= 0 && edgeMapVector[i][j + 1] ==
255 && edgeMapVector[i - 1][j + 1] == 255 && edgeMapVector[i + 1][j + 1] != 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tDR_191" << endl;
                        return "DR";//3
                }
                else if (j + 1 < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() && i + 1
< edgeMapVector.size() && edgeMapVector[i - 1][j + 1] != 255 && edgeMapVector[i + 1][j] == 255
&& edgeMapVector[i][j + 1] == 255)
                {
                        //cout << to_string(i) << "\t" << to_string(j) << "\tUR_192" << endl;
                        return "UR";//3
```

```
		}
		else if (j + 1 < edgeMapVector[0].size() && i + 1 < edgeMapVector.size() && i - 1
< edgeMapVector.size() && edgeMapVector[i + 1][j + 1] != 255 && edgeMapVector[i - 1][j] == 255
&& edgeMapVector[i][j + 1] == 255)
		{
			//cout << to_string(i) << "\t" << to_string(j) << "\tDR_193" << endl;
			return "DR";//3
		}
		else if (j - 1 < edgeMapVector[0].size() && i + 1 < edgeMapVector.size() &&
edgeMapVector[i + 1][j - 1] != 255)
		{
			direction = 'L';
			//cout << to_string(i) << "\t" << to_string(j) << "\tDL_194" << endl;
			return "DL";//1
		}
		else if (j - 1 < edgeMapVector[0].size() && i - 1 < edgeMapVector.size() &&
edgeMapVector[i - 1][j - 1] != 255)
		{
			direction = 'L';
			//cout << to_string(i) << "\t" << to_string(j) << "\tUL_195" << endl;
			return "UL";//1
		}
		else if (j - 1 < edgeMapVector[0].size() && i < edgeMapVector.size() &&
edgeMapVector[i][j - 1] != 255)
		{
			direction = 'L';
			/*insertCornerPoints(pid, i, j - 1);*/
			//cout << to_string(i) << "\t" << to_string(j) << "\tL_196" << endl;
			return "L";//1
		}
		else
			return "0";
		break;
	default:
		return "0";
	}
}

void CannyThreshold(int, void*)
{
	/// Reduce noise with a kernel 3x3
	Mat blurImage;
	blur(src_gray, blurImage, Size(3, 3));

	Mat mask;

	/// Canny detector
	Canny(blurImage, detected_edges, lowThreshold, lowThreshold*ratios, kernel_size);
	lowThreshold = 70; // lower end
	Canny(blurImage, mask, lowThreshold, lowThreshold*ratios, kernel_size, true);
	for (int i = 0; i < mask.rows; i++)
	{
		for (int j = 0; j < mask.cols; j++)
		{
			if (mask.at<uchar>(i, j) == 255)
				mask.at<uchar>(i, j) = 0;
			else
```

```
                                        mask.at<uchar>(i, j) = 255;
                        }
                }
                /// Using Canny's output as a mask, we display our result
                dst = Scalar::all(0);

                detected_edges.copyTo(dst, mask);

}

void colorEnhancement()
{
                new_image = Mat::zeros(image.size(), image.type());

                /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
                for (int y = 0; y < image.rows; y++)
                {
                        for (int x = 0; x < image.cols; x++)
                        {
                                for (int c = 0; c < 3; c++)
                                {
                                        new_image.at<Vec3b>(y, x)[c] =
                                                saturate_cast<uchar>(alpha*(image.at<Vec3b>(y, x)[c]) +
beta);
                                }
                        }
                }

}

void MyFilledCircle(Mat &img, Point center)
{
                int thickness = -1;
                int lineType = 8;

                circle(img,
                        center,
                        5,
                        Scalar(0, 255, 255),
                        thickness,
                        lineType);
}

void MyFilledCircle2(Mat &img, Point center)
{
                int thickness = -1;
                int lineType = 8;

                circle(img,
                        center,
                        5,
                        Scalar(255,255,0),
                        thickness,
                        lineType);
}

void particalPathMarker(Mat &img, Point center)
```

```
{
	int thickness = 1;
	int lineType = 8;

	circle(img,
		center,
		1,
		Scalar(0, 255, 0),
		thickness,
		lineType);
}

void writeMatToFile(cv::Mat& m, const char* filename)
{
	try
	{
		cout << "File Write Start" << endl;
		//   //ofstream fout(filename);
		  //edgeMapFile.crea
		edgeMapFile.open("edgeMap.txt");
		////edgeMapFile<<XR<<"_"<<YR<<"_"<<ZR<<endl;
		//cout<<m.channels()<<endl;

		if (!edgeMapFile)
		{
			cout << "File Not Opened" << endl;  return;
		}

		for (int i = 0; i < detected_edges.rows; i++)
		{
			for (int j = 0; j < detected_edges.cols; j++)
			{
				//fout<<m.at<float>(i,j)<<"\t";
				edgeMapFile << (float)detected_edges.at<uchar>(i, j) << "\t";
				tempVector.push_back((int)detected_edges.at<uchar>(i, j));
				//cout<<"hello"<<"\t";
			}
			edgeMapFile << endl;
			edgeMapVector.push_back(tempVector);
			tempVector.clear();
		}
		edgeMapFile.close();
		cout << "File Write Done" << endl;
		cout << "rows " << edgeMapVector.size() << endl;
		cout << "col " << edgeMapVector[0].size() << endl;
	}
	catch (Exception e)
	{
		cout << e.msg << endl;
	}
}

void FileToMatObject(cv::Mat& m, cv::Mat& t, const char* filename)
{
	ifstream infile("edgeMap.txt");
	string s;
	int countLines = 0;
```

```
	vector<string> tokens;
	/*while (infile >>s)
	{
		cout<<s<<endl;
	}*/
	for (std::string line; getline(infile, line); )
	{
		//cout<<line<<endl;
		istringstream iss(line);
		copy(istream_iterator<string>(iss),
			istream_iterator<string>(),
			back_inserter(tokens));
		countLines++;
		//cout<<"Count of the lines "<<countLines<<endl;
	}
	//cout<<"Count of the lines "<<countLines<<endl;
	imageRows = countLines;
	cout << "Vector size " << tokens.size() << endl;
	imageColumns = tokens.size() / imageRows;
	int vectorCounter = 0;

	//Mat ut(imageRows,imageColumns,m.channels);
	//CV_Assert(t.depth() == CV_8U);
	Mat a(imageRows, imageColumns, CV_8UC1);

	for (int i = 0; i < imageRows; i++)
	{
		for (int j = 0; j < imageColumns; j++)
		{
			if (tokens.size() > vectorCounter)
			{
				a.at<uchar>(i, j) = stoi(tokens.at(vectorCounter));
			}
			vectorCounter++;
		}
	}
	cout << "done" << endl;
	//imshow("Intersections",a);
}

void FileToMatObject2(cv::Mat& m, cv::Mat& t, const char* filename)
{
	//ifstream infile("edgeMapRemove.txt");//eye remove unwanted parts
	ifstream infile("edgeMapRemoveRoad2.txt"); //road unwanted parts removed
	string s;
	int countLines = 0;
	vector<string> tokens;
	/*while (infile >>s)
	{
		cout<<s<<endl;
	}*/
	for (std::string line; getline(infile, line); )
	{
		//cout<<line<<endl;
		istringstream iss(line);
		copy(istream_iterator<string>(iss),
			istream_iterator<string>(),
```

```
                back_inserter(tokens));
        countLines++;
        //cout<<"Count of the lines 2 "<<countLines<<endl;
    }
    //cout<<"Count of the lines "<<countLines<<endl;
    imageRows = countLines;
    cout << "Vector size " << tokens.size() << endl;
    imageColumns = tokens.size() / imageRows;
    int vectorCounter = 0;

    //Mat ut(imageRows,imageColumns,m.channels);
    //CV_Assert(t.depth() == CV_8U);
    Mat b(imageRows, imageColumns, CV_8UC1);
    cout << "c count" << b.channels() << endl;
    for (int i = 0; i < imageRows; i++)
    {
        for (int j = 0; j < imageColumns; j++)
        {
            if (tokens.size() > vectorCounter)
            {
                b.at<uchar>(i, j) = stoi(tokens.at(vectorCounter));
            }
            vectorCounter++;
        }
    }
    cout << "done" << endl;
    imshow("After Remove", b);
}

void clearUnwantedEdges(Mat m)
{
    for (int i = 0; i < m.rows; i++)
    {
        for (int j = 0; j < m.cols; j++)
        {
            if ((int)m.at<uchar>(i, j) < 500)
            {
                m.at<uchar>(i, j) = 0;
            }
        }
    }

    /*imshow("afterRemove",m);*/
}

void vectorPrint()
{
    cout << "col" << detected_edges.cols << endl;
    cout << "rows" << detected_edges.rows << endl;
    cout << "capacity" << edgeMapVector.size() << endl;
    cout << "capacity" << edgeMapVector[0].size() << endl;
}

static int callback(void *NotUsed, int argc, char **argv, char **azColName)
{
    int i;
    for (i = 0; i < argc; i++)
```

```
	{
		cout << azColName[i] << "\t" << argv[i] << "\t";
		//cout << azColName[i] << "\t" << argv[i] << "\t";

	}
	cout << endl;
	return 0;
}

// 1 true 0 false -1 invalid
static int callbackCheckSimilarMovements(void *NotUsed, int argc, char **argv, char **azColName)
{
	/*int i;
	for (i = 0; i < argc; i++)
	{
		cout << azColName[i] << "\t" << argv[i] << "\t";
	}
	cout << endl;*/

	if (argc == 1)
	{
		if (atoi(argv[0]) >= 5)
		{
			similarMovementRetValue = 1;
		}
		else
		{
			similarMovementRetValue = 0;
		}
	}
	else
	{
		similarMovementRetValue = -1;
	}

	return 0;
}

static int callBackShowIntersections(void *NotUsed, int argc, char **argv, char **azColName)
{

	MyFilledCircle(intersection_image, Point(atoi(argv[1]), atoi(argv[0])));

	return 0;
}

static int callBackShowCornerPoints(void *NotUsed, int argc, char **argv, char **azColName)
{

	MyFilledCircle2(intersection_image, Point(atoi(argv[1]), atoi(argv[0])));

	return 0;
}

static int callBackShowParticalPaths(void *NotUsed, int argc, char **argv, char **azColName)
{
	particalPathMarker(intersection_image, Point(atoi(argv[1]), atoi(argv[0])));
```

```
	return 0;
}

static int callBackSaveParticalCoordinates(void *NotUsed, int argc, char **argv, char **azColName)
{
	pathCoordinatesX.push_back(atoi(argv[0]));
	pathCoordinatesY.push_back(atoi(argv[1]));
	return 0;
}

void emptyTable()
{
	string sqlQuery = "delete from data_log_tbl;";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	sqlQuery = "UPDATE SQLITE_SEQUENCE SET SEQ = 0 WHERE NAME =
'data_log_tbl';";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	sqlQuery = "DELETE FROM partical_direction_tbl;";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	sqlQuery = "UPDATE SQLITE_SEQUENCE SET SEQ = 0 WHERE NAME =
'partical_direction_tbl';";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	sqlQuery = "DELETE FROM corner_points_tbl;";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	sqlQuery = "UPDATE SQLITE_SEQUENCE SET SEQ = 0 WHERE NAME =
'corner_points_tbl';";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "Table Reset\n");
	}
}

//void insertData(int particalID,int source_x,int source_y,int dest_x,int dest_y)
void insertData(int particalID, int source_x, int source_y, int dest_x, int dest_y, string dir)
{
	//string name = "ajith";
	string sqlQuery = "insert into data_log_tbl (partical_id,source_x,source_y,dest_x,dest_y)
values ('" + to_string(particalID) + "','" + to_string(source_x) + "','" + to_string(source_y) + "','" +
to_string(dest_x) + "','" + to_string(dest_y) + "');";
```

```
	//string sqlQuery = "insert into data_log2_tbl
(partical_id,source_x,source_y,dest_x,dest_y,direction) values ('"+to_string(particalID)+"','"+
to_string(source_x)+"','"+ to_string(source_y)+"','"+ to_string(dest_x)+"','"+
to_string(dest_y)+"','"+dir+"');";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		//fprintf(stdout, "inserted\n");
	}
}

void insertParticalDirection(int particalID, string dir)
{
	string sqlQuery = "insert into partical_direction_tbl (partical_id,direction) values ('" +
to_string(particalID) + "','" + dir + "');";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		//fprintf(stdout, "inserted\n");
	}
}

void insertCornerPoints(int particalID, int x, int y)
{
	string sqlQuery = "insert into corner_points_tbl (partical_id,x,y) values ('" +
to_string(particalID) + "','" + to_string(x) + "','" + to_string(y) + "');";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		//fprintf(stdout, "inserted\n");
	}
}

void selectAll()
{
	sql = "select * from data_log_tbl;";
```

```
	rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "select\n");
	}
}

void checkSimilarMovements(int dest_x, int dest_y)
{
	string sqlQuery = "select count(id) as rowCount from data_log_tbl where dest_x =" +
to_string(dest_x) + " and dest_y=" + to_string(dest_y) + ";";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callbackCheckSimilarMovements, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		//fprintf(stdout, "select\n");
	}
}

void test(string name)
{
	//insertData(name);
	selectAll();
}

void movePartical(int pID, int x,int y, char particalDirection)
//void movePartical(int pID, int startingPoint, char particalDirection)
{
	try
	{

		int i = x;
		int j = y;
		char* retValue = "0";
		int source_x = 0;
		int source_y = 0;
		int dest_x = 0;
		int dest_y = 0;
		int particalID = pID;
		int counter = 0;
		bool executionStop = false;
		int tempVal = 0;
		direction = particalDirection;
```

```
if (direction == 'R')
{
	i = x;
	j = y;
	retValue = isValidMoveAvailable(pID, i, j, direction);
}
else if (direction == 'L')
{
	/*i = startingPoint;
	j = edgeMapVector[0].size() - 1;*/
	i = x;
	j = y;
	retValue = isValidMoveAvailable(pID, i, j, direction);
}
else if (direction == 'U')
{
	/*i = edgeMapVector.size() - 1;
	j = startingPoint;*/
	i = x;
	j = y;
	retValue = isValidMoveAvailable(pID, i, j, direction);
}
else if (direction == 'D')
{
	/*i = 0;
	j = startingPoint;*/
	i = x;
	j = y;
	retValue = isValidMoveAvailable(pID, i, j, direction);
}


while (retValue != "0" && executionStop != true)
{
	counter++;
	if (retValue == "U")
	{
		source_x = i;
		source_y = j;
		i = i - 1;
		dest_x = i;
		dest_y = j;
		//insertData(particalID, source_x, source_y, dest_x, dest_y);
		insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

		if (direction == 'R')
		{
			retValue = isValidMoveAvailable(pID, i, j, direction);
		}
		else if (direction == 'L')
		{
			retValue = isValidMoveAvailable(pID, i, j, direction);
		}
		else if (direction == 'U')
		{
			retValue = isValidMoveAvailable(pID, i, j, direction);
```

```
                }
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "D")
        {
                source_x = i;
                source_y = j;
                i = i + 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "L")
        {
                source_x = i;
                source_y = j;
                j = j - 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
```

```
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "R")
        {
                source_x = i;
                source_y = j;
                j = j + 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "UL")
        {
                source_x = i;
                source_y = j;
                i = i - 1;
                j = j - 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
```

```
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "UR")
        {
                source_x = i;
                source_y = j;
                i = i - 1;
                j = j + 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'D')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
        }
        else if (retValue == "DL")
        {
                source_x = i;
                source_y = j;
                i = i + 1;
                j = j - 1;
                dest_x = i;
                dest_y = j;
                //insertData(particalID, source_x, source_y, dest_x, dest_y);
                insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                if (direction == 'R')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'L')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
                }
                else if (direction == 'U')
                {
                        retValue = isValidMoveAvailable(pID, i, j, direction);
```

```
                        }
                        else if (direction == 'D')
                        {
                                retValue = isValidMoveAvailable(pID, i, j, direction);
                        }
                }
                else if (retValue == "DR")
                {
                        source_x = i;
                        source_y = j;
                        i = i + 1;
                        j = j + 1;
                        dest_x = i;
                        dest_y = j;
                        //insertData(particalID, source_x, source_y, dest_x, dest_y);
                        insertData(particalID, source_x, source_y, dest_x, dest_y,
retValue);

                        if (direction == 'R')
                        {
                                retValue = isValidMoveAvailable(pID, i, j, direction);
                        }
                        else if (direction == 'L')
                        {
                                retValue = isValidMoveAvailable(pID, i, j, direction);
                        }
                        else if (direction == 'U')
                        {
                                retValue = isValidMoveAvailable(pID, i, j, direction);
                        }
                        else if (direction == 'D')
                        {
                                retValue = isValidMoveAvailable(pID, i, j, direction);
                        }
                }

                checkSimilarMovements(dest_x, dest_y);

                if (similarMovementRetValue == 1 || similarMovementRetValue == -1)
                {
                        executionStop = true;
                }
        }

        cout << "Partical Movement Done" << endl;

    }
    catch (Exception e)
    {
        cout << e.msg << endl;
    }

}

void showIntersections()
{
```

```
	string sqlQuery = "select dest_x, dest_y from data_log_tbl group by dest_x, dest_y having
count(distinct partical_id) >=3 ;";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callBackShowIntersections, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "select\n");
	}
}

void showCornerPoints()
{
	string sqlQuery = "select distinct x, y from corner_points_tbl;";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callBackShowCornerPoints, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "select\n");
	}
}

void showParticalPaths(int id)
{
	string sqlQuery = "select dest_x, dest_y from data_log_tbl where partical_id = " +
to_string(id) + ";";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callBackShowParticalPaths, 0, &zErrMsg);

	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "select\n");
	}
}

void getParticalPathCoordinates(int id)
{
	string sqlQuery = "select dest_x,dest_y from data_log_tbl where partical_id = " + to_string(id)
+ ";";

	rc = sqlite3_exec(db, sqlQuery.c_str(), callBackSaveParticalCoordinates, 0, &zErrMsg);
```

```
	if (rc != SQLITE_OK)
	{
		fprintf(stderr, "SQL Error %s\n", zErrMsg);
		sqlite3_free(zErrMsg);
	}
	else
	{
		fprintf(stdout, "select\n");
	}
}

void writeParticalPathToFile(int id)
{
	try
	{
		ifstream infile("edgeMap.txt");
		string s;
		int countLines = 0;
		vector<string> tokens;

		for (std::string line; getline(infile, line); )
		{
			//cout<<line<<endl;
			istringstream iss(line);
			copy(istream_iterator<string>(iss),
				istream_iterator<string>(),
				back_inserter(tokens));
			countLines++;
		}
		imageRows = countLines;
		imageColumns = tokens.size() / imageRows;
		int vectorCounter = 0;

		Mat a(imageRows, imageColumns, CV_8UC1);

		for (int i = 0; i < imageRows; i++)
		{
			for (int j = 0; j < imageColumns; j++)
			{
				if (tokens.size() > vectorCounter)
				{
					a.at<uchar>(i, j) = stoi(tokens.at(vectorCounter));
				}
				vectorCounter++;
			}
		}

		while (!pathCoordinatesX.empty() && !pathCoordinatesY.empty())
		{
			int x = pathCoordinatesX.back();
			int y = pathCoordinatesY.back();

			a.at<uchar>(x, y) = id;
			pathCoordinatesX.pop_back();
			pathCoordinatesY.pop_back();
		}
```

```
		cout << "File Write Start" << endl;
		edgeMapFile.open("edgeMap_" + to_string(id) + ".txt");

		if (!edgeMapFile)
		{
			cout << "File Not Opened" << endl;  return;
		}

		for (int i = 0; i < a.rows; i++)
		{
			for (int j = 0; j < a.cols; j++)
			{
				edgeMapFile << (float)a.at<uchar>(i, j) << "\t";
			}
			edgeMapFile << endl;
		}
		edgeMapFile.close();
		cout << "File Write Done" << endl;
	}
	catch (Exception e)
	{
		cout << e.msg << endl;
	}
}

void findSeedPoints()
{
	int i = 0;
	int j = 0;
	vector<int> temp;

#pragma region Left to right

	//left seed points only rows j=0
	i = 0;
	j = 0;

	for (i; i < edgeMapVector.size(); i++)
	{
		if (edgeMapVector[i][j] == 255)
		{
			temp.push_back(i);
			cout << "L" << i << endl;
		}
	}


	while (!temp.empty())
	{
		int v1 = temp.back();
		temp.pop_back();
		vector<int> t;

		if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
		{
			t.push_back(v1 - 1);
```

```
                t.push_back(0);
                leftSeedPoints.push_back(t);
        }
        else
        {
                t.push_back(v1);
                t.push_back(0);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
        {
                t.push_back(v1 + 1);
                t.push_back(0);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1 + 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(0);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1);
        }
}
temp.clear();

//L1

i = 0;
j = (int)edgeMapVector[0].size()/3;

for (i; i < edgeMapVector.size(); i++)
{
        if (edgeMapVector[i][j] == 255)
        {
                temp.push_back(i);
                cout << "L_1" << i << endl;
        }
}

while (!temp.empty())
{
        int v1 = temp.back();
        temp.pop_back();
        vector<int> t;

        if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
        {
                t.push_back(v1 - 1);
                t.push_back(j);
                leftSeedPoints.push_back(t);
        }
        else
        {
                t.push_back(v1);
```

```
                t.push_back(j);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
        {
                t.push_back(v1 + 1);
                t.push_back(j);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1 + 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1);
        }
}
temp.clear();

//L2
i = 0;
j = (int)edgeMapVector[0].size() / 1.5;

for (i; i < edgeMapVector.size(); i++)
{
        if (edgeMapVector[i][j] == 255)
        {
                temp.push_back(i);
                cout << "L_2" << i << endl;
        }
}

while (!temp.empty())
{
        int v1 = temp.back();
        temp.pop_back();
        vector<int> t;

        if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
        {
                t.push_back(v1 - 1);
                t.push_back(j);
                leftSeedPoints.push_back(t);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                leftSeedPoints.push_back(t);
                //leftSeedPoints.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
        {
```

```
                    t.push_back(v1 + 1);
                    t.push_back(j);
                    leftSeedPoints.push_back(t);
                    //leftSeedPoints.push_back(v1 + 1);
                }
                else
                {
                    t.push_back(v1);
                    t.push_back(j);
                    leftSeedPoints.push_back(t);
                    //leftSeedPoints.push_back(v1);
                }
            }
            temp.clear();

#pragma endregion

#pragma region Right to Left
            //right seed points only rows j=last pixel
            i = 0;
            j = edgeMapVector[0].size() - 1;

            for (i; i < edgeMapVector.size(); i++)
            {
                if (edgeMapVector[i][j] == 255)
                {
                    temp.push_back(i);
                    cout << "R" << i << endl;
                }
            }


            while (!temp.empty())
            {
                int v1 = temp.back();
                temp.pop_back();
                vector<int> t;

                if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
                {
                    t.push_back(v1 - 1);
                    t.push_back(j);
                    rightSeedPoints.push_back(t);
                    //rightSeedPoints.push_back(v1 - 1);
                }
                else
                {
                    t.push_back(v1);
                    t.push_back(j);
                    rightSeedPoints.push_back(t);
                    //rightSeedPoints.push_back(v1);
                }

                if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
                {
                    t.push_back(v1 + 1);
                    t.push_back(j);
```

```
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1 + 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1);
        }
}
temp.clear();

//R1
i = 0;
j = (int)(edgeMapVector[0].size() - 1)/4;

for (i; i < edgeMapVector.size(); i++)
{
        if (edgeMapVector[i][j] == 255)
        {
                temp.push_back(i);
                cout << "R_1" << i << endl;
        }
}


while (!temp.empty())
{
        int v1 = temp.back();
        temp.pop_back();
        vector<int> t;

        if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
        {
                t.push_back(v1 - 1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1 - 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
        {
                t.push_back(v1 + 1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1 + 1);
        }
        else
        {
```

```
                t.push_back(v1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1);
        }
}
temp.clear();

//R2

i = 0;
j = (int)(edgeMapVector[0].size() - 1) / 2;

for (i; i < edgeMapVector.size(); i++)
{
        if (edgeMapVector[i][j] == 255)
        {
                temp.push_back(i);
                cout << "R_2" << i << endl;
        }
}


while (!temp.empty())
{
        int v1 = temp.back();
        temp.pop_back();
        vector<int> t;

        if (v1 - 1 < edgeMapVector.size() && v1 - 1 > 0)
        {
                t.push_back(v1 - 1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1 - 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector.size() && v1 + 1 > 0)
        {
                t.push_back(v1 + 1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1 + 1);
        }
        else
        {
                t.push_back(v1);
                t.push_back(j);
                rightSeedPoints.push_back(t);
                //rightSeedPoints.push_back(v1);
```

```
			}
		}
		temp.clear();

#pragma endregion

#pragma region Up to down
		//up seed point only cols i=0
		i = 0;
		j = 0;

		for (j; j < edgeMapVector[0].size(); j++)
		{
			if (edgeMapVector[i][j] == 255)
			{
				temp.push_back(j);
				cout << "U" << j << endl;
			}
		}


		while (!temp.empty())
		{
			int v1 = temp.back();
			temp.pop_back();
			vector<int> t;

			if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
			{
				t.push_back(i);
				t.push_back(v1 - 1);
				upSeedPoints.push_back(t);
				//upSeedPoints.push_back(v1 - 1);
			}
			else
			{
				t.push_back(i);
				t.push_back(v1);
				upSeedPoints.push_back(t);
				//upSeedPoints.push_back(v1);
			}

			if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
			{
				t.push_back(i);
				t.push_back(v1 + 1);
				upSeedPoints.push_back(t);
				//upSeedPoints.push_back(v1 + 1);
			}
			else
			{
				t.push_back(i);
				t.push_back(v1);
				upSeedPoints.push_back(t);
				//upSeedPoints.push_back(v1);
			}
		}
```

```
temp.clear();

//U1
i = (int) edgeMapVector.size()/3;
j = 0;

for (j; j < edgeMapVector[0].size(); j++)
{
	if (edgeMapVector[i][j] == 255)
	{
		temp.push_back(j);
		cout << "U_1" << j << endl;
	}
}


while (!temp.empty())
{
	int v1 = temp.back();
	temp.pop_back();
	vector<int> t;

	if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
	{
		t.push_back(i);
		t.push_back(v1 - 1);
		upSeedPoints.push_back(t);
		//upSeedPoints.push_back(v1 - 1);
	}
	else
	{
		t.push_back(i);
		t.push_back(v1);
		upSeedPoints.push_back(t);
		//upSeedPoints.push_back(v1);
	}

	if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
	{
		t.push_back(i);
		t.push_back(v1 + 1);
		upSeedPoints.push_back(t);
		//upSeedPoints.push_back(v1 + 1);
	}
	else
	{
		t.push_back(i);
		t.push_back(v1);
		upSeedPoints.push_back(t);
		//upSeedPoints.push_back(v1);
	}
}
temp.clear();

//U2

i = (int)edgeMapVector.size() / 1.5;
```

```
	j = 0;

	for (j; j < edgeMapVector[0].size(); j++)
	{
		if (edgeMapVector[i][j] == 255)
		{
			temp.push_back(j);
			cout << "U_2" << j << endl;
		}
	}


	while (!temp.empty())
	{
		int v1 = temp.back();
		temp.pop_back();
		vector<int> t;

		if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
		{
			t.push_back(i);
			t.push_back(v1 - 1);
			upSeedPoints.push_back(t);
			//upSeedPoints.push_back(v1 - 1);
		}
		else
		{
			t.push_back(i);
			t.push_back(v1);
			upSeedPoints.push_back(t);
			//upSeedPoints.push_back(v1);
		}

		if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
		{
			t.push_back(i);
			t.push_back(v1 + 1);
			upSeedPoints.push_back(t);
			//upSeedPoints.push_back(v1 + 1);
		}
		else
		{
			t.push_back(i);
			t.push_back(v1);
			upSeedPoints.push_back(t);
			//upSeedPoints.push_back(v1);
		}
	}
	temp.clear();

#pragma endregion

#pragma region Down to Up

	//down seed point only cols i= last pixel
	i = edgeMapVector.size() - 1;
	j = 0;
```

```
for (j; j < edgeMapVector[0].size(); j++)
{
	if (edgeMapVector[i][j] == 255)
	{
		temp.push_back(j);
		cout << "D" << j << endl;
	}
}


while (!temp.empty())
{
	int v1 = temp.back();
	temp.pop_back();
	vector<int> t;

	if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
	{
		t.push_back(i);
		t.push_back(v1 - 1);
		downSeedpoint.push_back(t);
		//downSeedpoint.push_back(v1 - 1);
	}
	else
	{
		t.push_back(i);
		t.push_back(v1);
		downSeedpoint.push_back(t);
		//downSeedpoint.push_back(v1);
	}

	if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
	{
		t.push_back(i);
		t.push_back(v1 + 1);
		downSeedpoint.push_back(t);
		//downSeedpoint.push_back(v1 + 1);
	}
	else
	{
		t.push_back(i);
		t.push_back(v1);
		downSeedpoint.push_back(t);
		//downSeedpoint.push_back(v1);
	}
}
temp.clear();

//D1
i = (int) (edgeMapVector.size() - 1)/4;
j = 0;

for (j; j < edgeMapVector[0].size(); j++)
{
	if (edgeMapVector[i][j] == 255)
	{
```

```
                temp.push_back(j);
                cout << "D_1" << j << endl;
        }
}


while (!temp.empty())
{
        int v1 = temp.back();
        temp.pop_back();
        vector<int> t;

        if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
        {
                t.push_back(i);
                t.push_back(v1 - 1);
                downSeedpoint.push_back(t);
                //downSeedpoint.push_back(v1 - 1);
        }
        else
        {
                t.push_back(i);
                t.push_back(v1);
                downSeedpoint.push_back(t);
                //downSeedpoint.push_back(v1);
        }

        if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
        {
                t.push_back(i);
                t.push_back(v1 + 1);
                downSeedpoint.push_back(t);
                //downSeedpoint.push_back(v1 + 1);
        }
        else
        {
                t.push_back(i);
                t.push_back(v1);
                downSeedpoint.push_back(t);
                //downSeedpoint.push_back(v1);
        }
}
temp.clear();

//D2
i = (int)(edgeMapVector.size() - 1) / 2;
j = 0;

for (j; j < edgeMapVector[0].size(); j++)
{
        if (edgeMapVector[i][j] == 255)
        {
                temp.push_back(j);
                cout << "D_2" << j << endl;
        }
}
```

```
		while (!temp.empty())
		{
			int v1 = temp.back();
			temp.pop_back();
			vector<int> t;

			if (v1 - 1 < edgeMapVector[0].size() && v1 - 1 > 0)
			{
				t.push_back(i);
				t.push_back(v1 - 1);
				downSeedpoint.push_back(t);
				//downSeedpoint.push_back(v1 - 1);
			}
			else
			{
				t.push_back(i);
				t.push_back(v1);
				downSeedpoint.push_back(t);
				//downSeedpoint.push_back(v1);
			}

			if (v1 + 1 < edgeMapVector[0].size() && v1 + 1 > 0)
			{
				t.push_back(i);
				t.push_back(v1 + 1);
				downSeedpoint.push_back(t);
				//downSeedpoint.push_back(v1 + 1);
			}
			else
			{
				t.push_back(i);
				t.push_back(v1);
				downSeedpoint.push_back(t);
				//downSeedpoint.push_back(v1);
			}
		}
		temp.clear();

#pragma endregion

}

int main(int argc, char** argv)
{
		clock_t tStart = clock();
		/*char* direction = isValidMoveAvailable(1, 2, "ER");
		cout << direction << endl;*/
		colorEnhancement();
		/// Load an image
		src = new_image;


		if (!src.data)
		{
			return -1;
		}
```

```
	/// Create a matrix of the same type and size as src (for dst)
	dst.create(src.size(), src.type());

	/// Convert the image to grayscale
	cvtColor(src, src_gray, CV_BGR2GRAY);

	/// Create a window
	namedWindow(window_name, CV_WINDOW_AUTOSIZE);

	/// Create a Trackbar for user to enter threshold

	createTrackbar("Min Threshold: ", window_name, &lowThreshold, max_lowThreshold,
CannyThreshold);

	/// Show the image

	CannyThreshold(0, 0);

	writeMatToFile(dst, "edgeMap.txt");
	findSeedPoints();
	intersection_image =  dst; //show partical paths


	//open DB
	rc = sqlite3_open("partical_method_db.db", &db);
	if (rc)
	{
		fprintf(stderr, "Cant open DB %s\n", sqlite3_errmsg(db));
		return 0;
	}
	else
	{
		fprintf(stdout, "DB opened\n");
	}

	//insert some SQL
	//insertData("ajith");
	//selectAll();

	emptyTable();

	direction = 'R';
	int seedpointCounter = 1;
	while (!leftSeedPoints.empty())
	{
		direction = 'R';
		vector<int> tempVal;
		tempVal = leftSeedPoints.back();
		if (tempVal.size() == 4)
		{
			cout << "L " << tempVal[2] << " " << tempVal[3] << endl;
			insertParticalDirection(seedpointCounter, "R");
			movePartical(seedpointCounter, tempVal[2], tempVal[3], direction);
			leftSeedPoints.pop_back();
			seedpointCounter++;
		}
```

```
		else
		{
			cout << "L " << tempVal[0] << " " << tempVal[1] << endl;
			insertParticalDirection(seedpointCounter, "R");
			movePartical(seedpointCounter, tempVal[0], tempVal[1], direction);
			leftSeedPoints.pop_back();
			seedpointCounter++;
		}

}


direction = 'L';

while (!rightSeedPoints.empty())
{
		direction = 'L';
		vector<int> tempVal;
		tempVal = rightSeedPoints.back();
		if (tempVal.size() == 4)
		{
			cout << "R " << tempVal[2] << " " << tempVal[3] << endl;
			insertParticalDirection(seedpointCounter, "L");
			movePartical(seedpointCounter, tempVal[2], tempVal[3], direction);
			rightSeedPoints.pop_back();
			seedpointCounter++;
		}
		else
		{
			cout << "R " << tempVal[0] << " " << tempVal[1] << endl;
			insertParticalDirection(seedpointCounter, "L");
			movePartical(seedpointCounter, tempVal[0], tempVal[1], direction);
			rightSeedPoints.pop_back();
			seedpointCounter++;
		}
}
//
//
direction = 'U';

while (!downSeedpoint.empty())
{
		direction = 'U';
		vector<int> tempVal;
		tempVal = downSeedpoint.back();
		if (tempVal.size() == 4)
		{
			cout << "D " << tempVal[2] << " " << tempVal[3] << endl;
			insertParticalDirection(seedpointCounter, "U");
			movePartical(seedpointCounter, tempVal[2], tempVal[3], direction);
			downSeedpoint.pop_back();
			seedpointCounter++;
		}
		else
		{
			cout << "D " << tempVal[0] << " " << tempVal[1] << endl;
			insertParticalDirection(seedpointCounter, "U");
```

```
                        movePartical(seedpointCounter, tempVal[0], tempVal[1], direction);
                        downSeedpoint.pop_back();
                        seedpointCounter++;
                }
        }
        //
        direction = 'D';
        //
        while (!upSeedPoints.empty())
        {
                direction = 'D';
                vector<int> tempVal;
                tempVal = upSeedPoints.back();
                if (tempVal.size() == 4)
                {
                        cout << "U " << tempVal[2] << " " << tempVal[3] << endl;
                        insertParticalDirection(seedpointCounter, "D");
                        movePartical(seedpointCounter, tempVal[2], tempVal[3], direction);
                        upSeedPoints.pop_back();
                        seedpointCounter++;
                }
                else
                {
                        cout << "U " << tempVal[0] << " " << tempVal[1] << endl;
                        insertParticalDirection(seedpointCounter, "D");
                        movePartical(seedpointCounter, tempVal[0], tempVal[1], direction);
                        upSeedPoints.pop_back();
                        seedpointCounter++;
                }

        }

        showIntersections();

        imshow("Intersections", intersection_image);
        imshow("Edge Map", detected_edges);

        sqlite3_close(db);

        printf("Time taken: %.2f s\n", ((double)(clock() - tStart) / CLOCKS_PER_SEC));
        //cout << "Number of Particals " << seedpointCounter-1 << endl;
        /// Wait until user exit program by pressing a key
        waitKey(0);

        return 0;
}
```