Autonomous Aerial Motion Planning using Image Processing/Computer Vision

Author

MUHAMMAD UMAIR

2011-NUST-MS PhD-Mts-24

Supervisor

DR UMAR SHAHBAZ KHAN

DEPARTMENT OF MECHATRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JULY, 2014

Autonomous Aerial Motion Planning using Image Processing/Computer Vision

Author

MUHAMMAD UMAIR

2011-NUST-MS PhD-Mts-24

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Mechatronics Engineering

Thesis Supervisor:

DR UMAR SHAHBAZ KHAN

Thesis Supervisor's Signature:_____

DEPARTMENT OF MECHATRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

JULY, 2014

Declaration

I certify that this research work titled "*Autonomous Aerial Motion Planning using Image Processing/Computer Vision"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Muhammad Umair

2011-NUST-Ms PhD-Mts-24

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

MUHAMMAD UMAIR

2011-NUST-MS PhD-Mts-24

Signature of Supervisor

Copyright Statement

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to pay special thanks to my supervisor Dr. Umar Shahbaz for his tremendous support and cooperation. Each time I got stuck in something, he came up with the solution. Without his help I wouldn't have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to express thanks to Dr.Rab Nawaz for his help throughout my thesis and also for Digital Image Processing and Computer Vision/Pattern Recognition which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to thank Dr. Kunwar Faraz Ahmed, Dr. Khurram Kamal and Dr. Umer Izhar for being on my thesis guidance and evaluation committee.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

Autonomous aerial motion planning in a dynamic environment is a complex task, whereas recent advancement in the theory of dynamics, kinematics and computer vision demonstrated a remarkable potential for futuristic autonomous aerial platforms. Research is focus on generating real time navigations based on ordinary camera images. It takes run time images as an input and estimate depth from a predefined tag (Region of Interest) within the image; finally after position localization it plans a path to the goal point.

Algorithm uses tag to update its orientation and to compute the distance. 2D images can be used to find the depth of a particular object in the image. Initially the shape of the marker is fixed and is pre-defined but as the research expands improvements will make the algorithm to work autonomously using everyday objects as tags.

Feature detection and extraction are well known techniques in the computer vision and image processing. Already many algorithms are developed in this field. Most of these algorithms are limited to certain frame of reference. Such as corner points, lines, binary features, boundary traces extraction or detection. Likewise derived algorithm has its own novelty element. Algorithm independently not only classifies a pre-defined tag from a single 2D camera image but also calculates distance to that object with accuracy. Designed algorithm makes use of an easy to use graphical user interface developed to demonstrate results. Graphical user interface is designed in the Matlab.

The research shows that single camera depth estimation can be achieved using polynomial curve fitting approach. For a said tag with known dimensions one can determine a fixed equation which can then be used to find any random distance. The approach is efficient and can effectively be applied to any indoor navigation or motion planning algorithm. Approach is not defendant upon expensive equipment and overall computational cost is also low. Ultimately the algorithm is capable of producing the output in just a few seconds with almost 91.84% accuracy.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

Autonomous aerial motion planning in a dynamic environment is a complex task, whereas recent advancements in the theory of dynamics, kinematics and computer vision demonstrated a remarkable potential for futuristic autonomous aerial platforms. Research is focused on generating real time navigations based on ordinary camera images. Computer vision being a scientific discipline acquires a real world data as an input and produces meaningful information.

1.1 Background, Scope and Motivation

The problem of motion planning in an environment cluttered by the obstacles can be addressed by three general approaches, namely cell decomposition methods, roadmap methods and artificial potential field methods [1]. Any algorithm which is complete returns a timely valid solution if it exists and failure in case if it is not feasible.

The objective of the autonomous motion planning is to automatically guide a robot in an environment filled with impediments. Obstacles may be static or in motion. Therefore, a great deal of concern is to first identify the obstacle and then plan the collision free path till the desired point. Besides just planning the optimal path and identification of obstacles, robot should be capable of estimating the depth to the obstacle and it should localize its position within the workspace.

Learning a depth only from an input camera feed at runtime has important applications in robotics. It requires consideration of global structure of the image [2]. A lot of research had been done for evaluating the depth of certain object using imagery data. Methods for finding the distance can be classified as active and passive. In active, some signals are transmitted to the object using different electronic sensors like ultra sound, laser beams, radio etc. and based on their feedback decision is taken; while passive rely on light [3].

1.2 Depth Estimation

1.2.1 Stereoscopic Depth Estimation

Technique used for recording and demonstrating stereoscopic images. It creates a delusion of depth using two pictures taken at slightly different position. There are two possibilities for obtaining stereoscopic images, first is by using stereoscopic camera specifically designed for the particular task, it contains two lens mounted within a single camera. Second option is to use a system based on two single lenses joined together. Both designs are shown in figure 1.1 below.

Figure 1.1: Stereoscopic camera (Left), system designed to acquire stereoscopic images (Right)

Distance is calculated from difference between the images taken from cameras, their focal length and distance between the positions of the lens [3]. It is assumed that lens should be horizontally aligned; pictures should be taken within the same environment and at the same instant. Any systematic or human error can be judged as shown in the figure 1.2

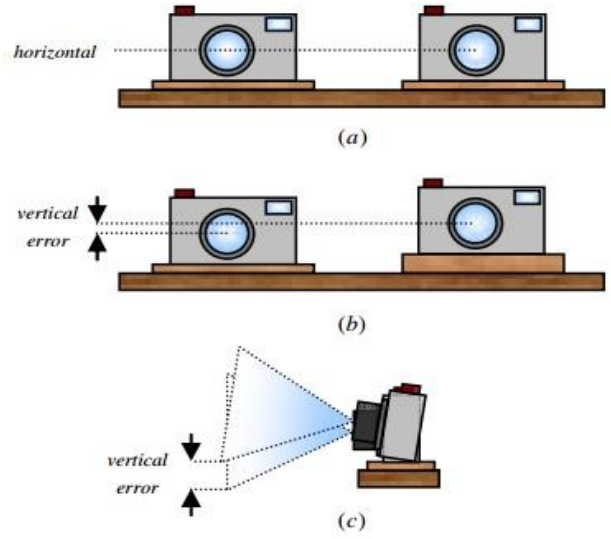

Figure 1.2: (a) Alignment of camera horizontally (b) and (c) vertical error [3]

Distance of a particular object in an image can be calculated by simple trigonometric identities shown in figure 1.3.

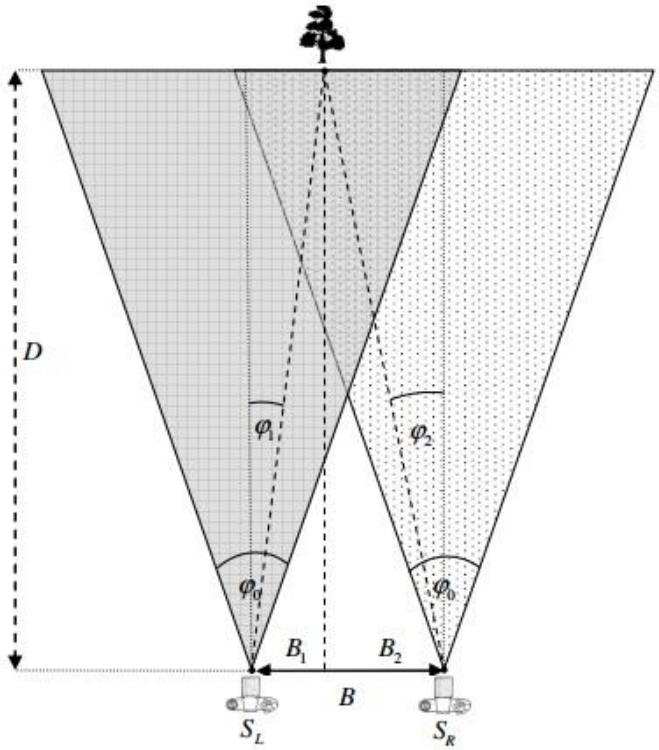

Figure 1.3: Stereoscopic image for distance estimation [3]

$$B = B1 + B2 = Dtan\varphi_1 + Dtan\varphi_2 \quad (1.1)$$

$$D = \frac{B}{tan\varphi_1 + tan\varphi_2} \quad (1.2)$$

Where ***'B'*** is the distance between two lenses and ***'φ'*** is their respective angle. Ratio of these two parameters ***'D'*** provides the estimated distance [3].

1.2.2 Markov Random Field

Most of the research work in 3D reconstruction is on binocular vision; beside just local features (geometric differences) other factors like haze, color, defocus, gradients; termed as monocular cues can also be utilize. Algorithm [1] based on Markov Random Field (MRF) uses monocular cues for depth calculation.

Image is divided into small patches. Depth of each patch is estimated using absolute depth and relative features. Feature that capture texture variations, texture gradients and haze is selected. Texture energy is calculated using Laws' mask [4], Haze being the low frequency element in the color it can be obtained by low averaging filter, finally texture gradient is the convolution of intensity channel with neighboring edges as shown in the figure 1.4

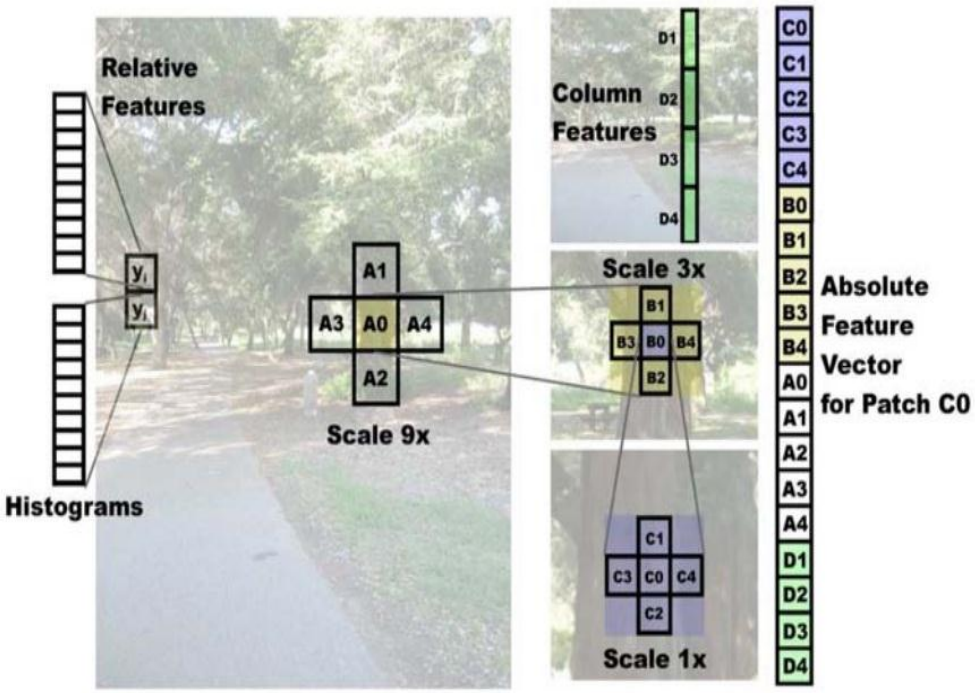

Figure 1.4: Absolute and relative depth features of a patch [5]

Depth of every patch depends upon the features of that patch and its neighboring patches. Therefore, probabilistic models can be used to drive the native relationship among the patches.

1.2.3 Depth Estimation in Digital Imaging

Depth of a particular object in the image can be obtained from several ways. It can also be derived through theory of optics. Beam of reflected light from an object after passing through convex lens must converge at a focal point.

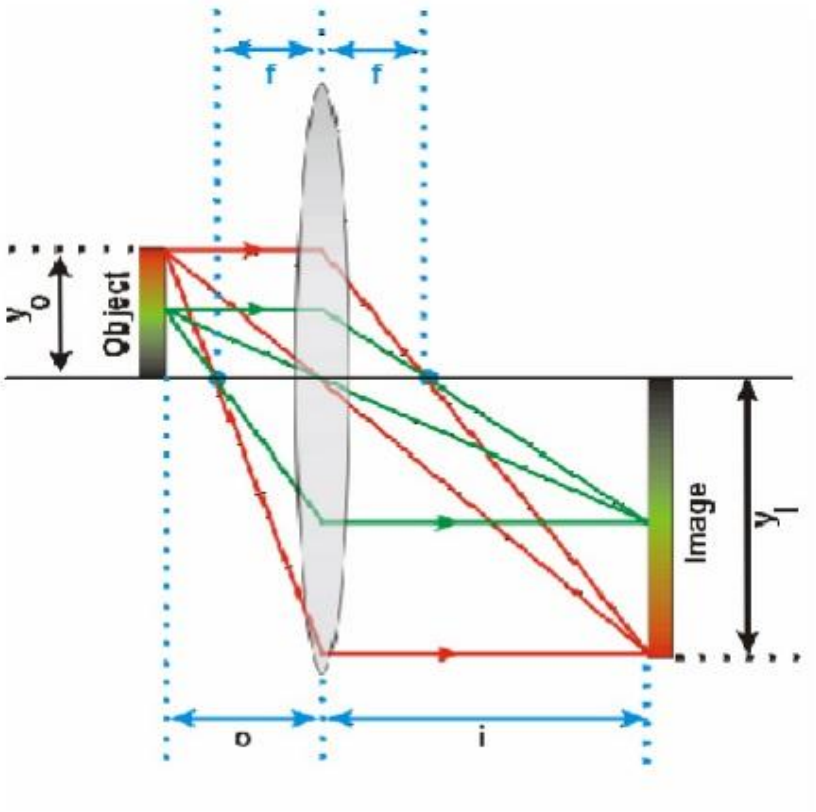

Figure 1.5: Theory of optics [6]

Magnification of the object is defined eq 1.3. Analyzing eq 1.3 if Y_i is equal to Y_o magnification will be one (1).

$$M = \frac{Y_i}{Y_o} \qquad (1.3)$$

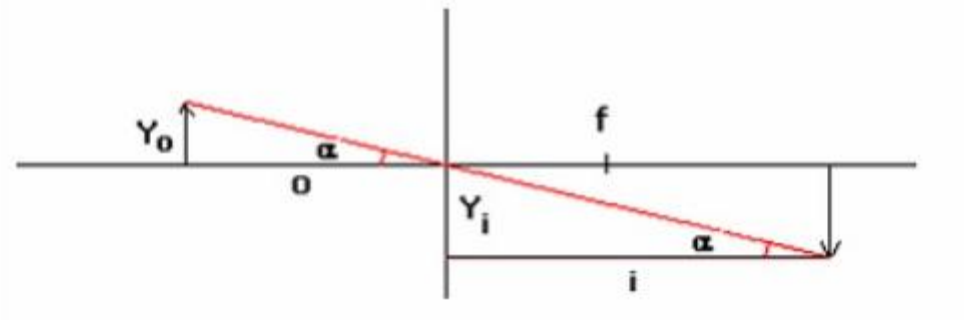

Figure 1.6: Depth estimation using theory of optics [6]

As shown in the figure 1.6, point of consideration is **'O'**. Simple trigonometric identities can be used to produce the desired result.

$$tan\ \alpha = \frac{Y_i}{i} \tag{1.4}$$

$$\alpha = tan^{-1}\left(\frac{Y_i}{i}\right) \tag{1.5}$$

Substitute and solve for 'O':

$$tan\ \alpha = \frac{Y_O}{O} \tag{1.6}$$

$$O = \left(\frac{Y_O}{tan\ \alpha}\right) \tag{1.7}$$

$$O = \frac{Y_O}{\tan[tan^{-1}(\frac{Y_i}{i})]} \tag{1.8}$$

$$O = \frac{iY_O}{Y_i} \tag{1.9}$$

1.2.4 Active Elements

Besides just computing the distance using pre-defined algorithms, depth can also be calculated using some electronic sensing aid. Sensors like sonar, IR can be used for the said purpose. These sensors generally transmit a particular signal which after reflecting back generates a feedback at the receiver's end. All such type of sensor based systems is called active equipment; but the aim of our research is to demonstrate the algorithm without utilizing any active element.

1.3 Object Classification and Detection

There are several techniques currently being used for classification of certain objects in the image. It not only depends upon the technique itself, some intrinsic properties of the object also play an important role for the identification.

Theoretical model for optical flow fields (velocity of pixels in the image sequence) can be used for detection of obstacle during runtime [7]. In order to perform detection of obstacle image is divided into two layers. Pixels which match the optical flow model are in first layer and which do not match corresponds to the second layer. Novelty of detecting the obstacles is to separate them from the ground floor in the image sequence. Moving obstacles are recognized based on the motion in the sequence of gathered images. Initially, model of the environment is extracted then theoretical optical filed model is compared with the actual video feed to observe the differences. Figure 1.7 shows the detected moving object, where highlighted red color depicts the contour of the detected moving obstacle.

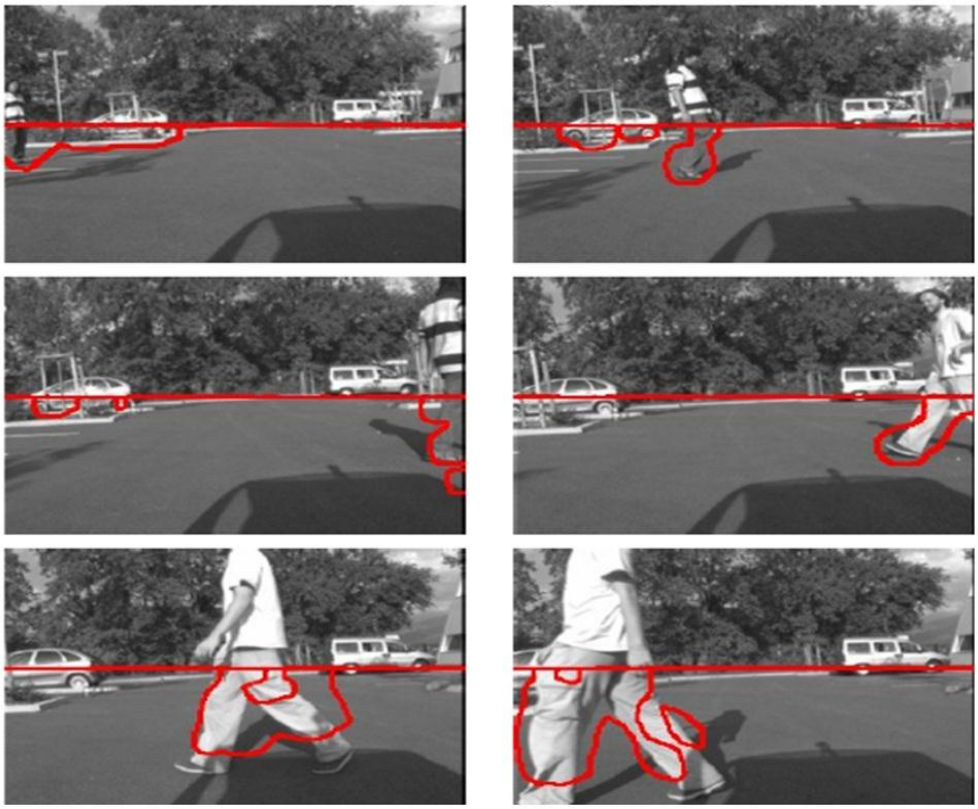

Figure 1.7: Detection of moving obstacles [7]

Once depth and classification of particular object is obtained successfully, position localization within the bounded work space can be worked out using simple mathematics. Autonomous aerial maneuvering has various applications (Robotics, Vehicles Automation, Military and Commercial use, and Industrial utilization). There are different vicinities where an autonomous indoor aerial surveillance is required, such as: surveillance of highly secured and sensitive areas, Transportation of different objects from one point to another.

CHAPTER 2: DEPTH ESTIMATION

Depth evaluation has significant importance in any type of autonomous motion planning algorithm. System should be capable of detecting any obstacle in its way and it should execute a timely decision to avoid collision. One of the basic problems while avoiding collision is to analyze the distance to the obstacle and feedback navigational controls.

Research is focused on depth evaluation using a single 2D image [8]. 2D images can be used for calculating distance to a particular object in the image. Designed algorithm uses marker of pre-defined shape to update position and orientation. Initially it is assumed that the shape of the marker is fixed but as the research expands improvements will make the algorithm to work autonomously for everyday routine objects as a marker. Scheme of research is not dependent on any additional sensor and it depends upon graceful statistical outcome. Valuation is divided into two phases, as shown in figure 2.1.

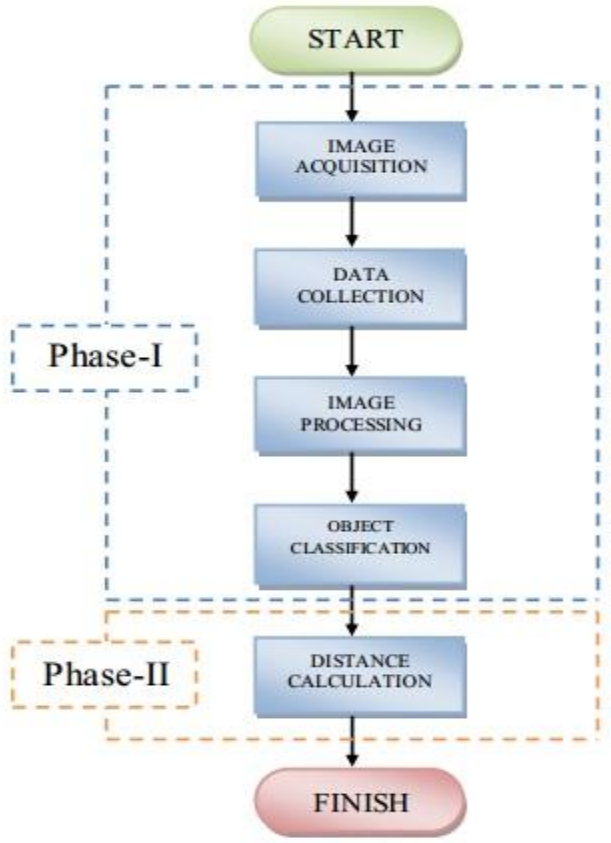

Figure 2.1: Flow chart representing general procedure (Phase-I and Phase-II)

Baseline for the said task is a natural phenomenon, suppose object of focus is at some distance ‘d’ from the lens. It is inversely proportional to the size of a particular object in the image, i.e. closer the object to the lens larger the size of the object in the image and vice versa. As shown in the figure 2.1, depth estimation can be classified as:

- Phase-I
    - Image acquisition
    - Data collection
    - Image processing
    - Object classification
- Phase-II
    - Distance calculation

Both image acquisition and data collection are to be considered as preprocessing for further application.

2.1 Image Acquisition

Everyday routine camera can be used for image acquisition. The process requires considerable cooperation from the user. The user must position the camera towards the marker, so that the marker falls within the field of view. It is assumed that same camera is used for a complete cycle. Any distortion at this stage can be neglected.

Region of interest (marker) should be clearly visible in the acquired images. It’s a good practice to get images in almost all possible type of lightening conditions except pitch dark. Images with some undesired noise margin will be processed in the image processing section. Therefore such images should not be neglected.

It is assumed that during the initial stage marker should be prominent in the image. Background should not have the same texture as that of the marker; there must be a certain observable difference between them. Once the camera calibration is done dataset of images will be attained in the data collection phase. Some sample images are shown in figure 2.2

Figure 2.2: Sample images during image acquisition process

As seen in the figure 2.1 region of interest is the rectangular shape object (marker), every image has multiple objects in it each having its own characteristics. Presently, our research work can only detect and classify the three basic colors i.e. red, blue and green. Therefore, currently there is a limited range of colors to be selected for the marker.

Camera input is a video feedback; frames are taken at runtime from this video and transferred to the integrated system for processing. Designed algorithm use ordinary mega pixel camera embedded in the laptops.

It is tested and demonstrated in Matlab, set of commands executes within a loop till required. Syntax used for image acquisition is:

```
% Declaration of intact camera characteristics and resolution for video input as vid
vid = videoinput('winvideo',1,'YUY2_640x480');

% Configuring feedback of vid as RGB
set(vid,'ReturnedColorSpace','RGB');

% Used to take snapshot of current frame
img=getsnapshot(vid);
```

2.2 Data Collection

Once the image acquisition is complete, a comprehensive data set of images is required for preprocessing. Usually this process can be referred as camera alignment. Images of marker are to be taken in varying environmental conditions.

Suppose the marker is at some distance ‘d’ from the camera. Where ‘d’ is equal to one feet. Minimum of five and a maximum of ten snapshots per feet are necessary for the orientation of the camera as well as of the environment. It’s a onetime measure. Camera alignment is only required once, when using the algorithm for the first time in any new location. Value of ‘d’ is incremented by one in each iteration i.e. let us consider following example pseudo run shown in figure 2.3

- Initialize
- At d=1 feet, take 5 images of marker, save all the images
- Increment d by 1 (d=d+1)
- Take 5 images and save all the images
- Repeat step 2 to 4 for d=1 to 10, and save the images accordingly in each iteration
- Finish

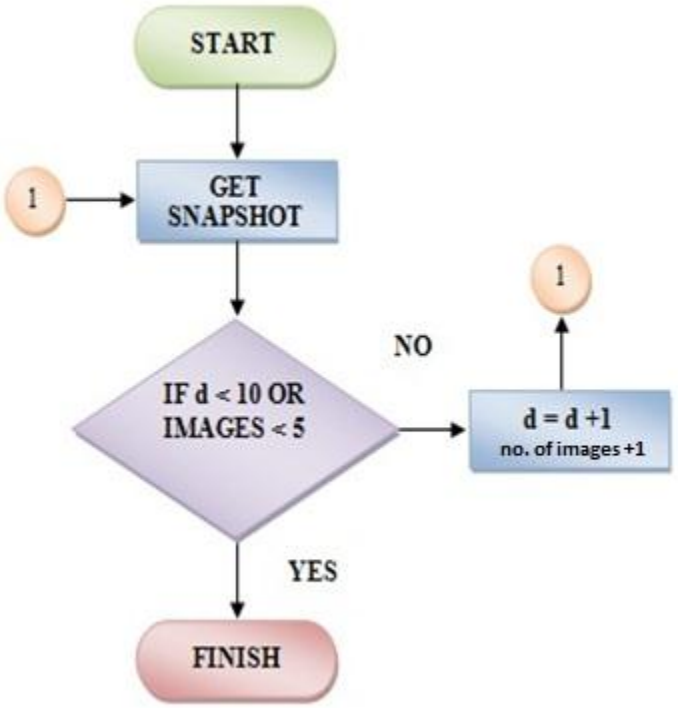

Figure 2.3: Flow chart for data collection

Gathered images needed to be sorted out with respect to the distance from which the images were taken. As ‘d’ ranges from one to ten therefore, up till now we should have ten data sets each containing five images. By looking at the captured images it can be observed that certain relationship between size of the marker in the images and value of ‘d’ exists. Images with less value of ‘d’ have large size of marker as compared to the one having higher value of ‘d’. These data sets are very useful for further processing and will be analyzed separately in the sections to come.

Single colored image is a combination of three basic image components (red, green and blue). Runtime image processing of an entire image is computational expensive and is a complex task as compared to a certain portion of the same image. Region of interest should be identified in all the images in order to process only selected portion of image. Every image is converted into its binary (zero or one) based on firm threshold. Images are converted into binary to identify each and every object in the image.

```
% Used to take snapshot of current frame
img=getsnapshot(vid);

% img is converted into its binay as Ibw
Ibw = im2bw(img);
```

At this stage all the necessary preprocessing is done, one should have data sets of images with respect to the distance from which they are taken and the binary counter part of each image. Figure 2.4 contains a sample data set.

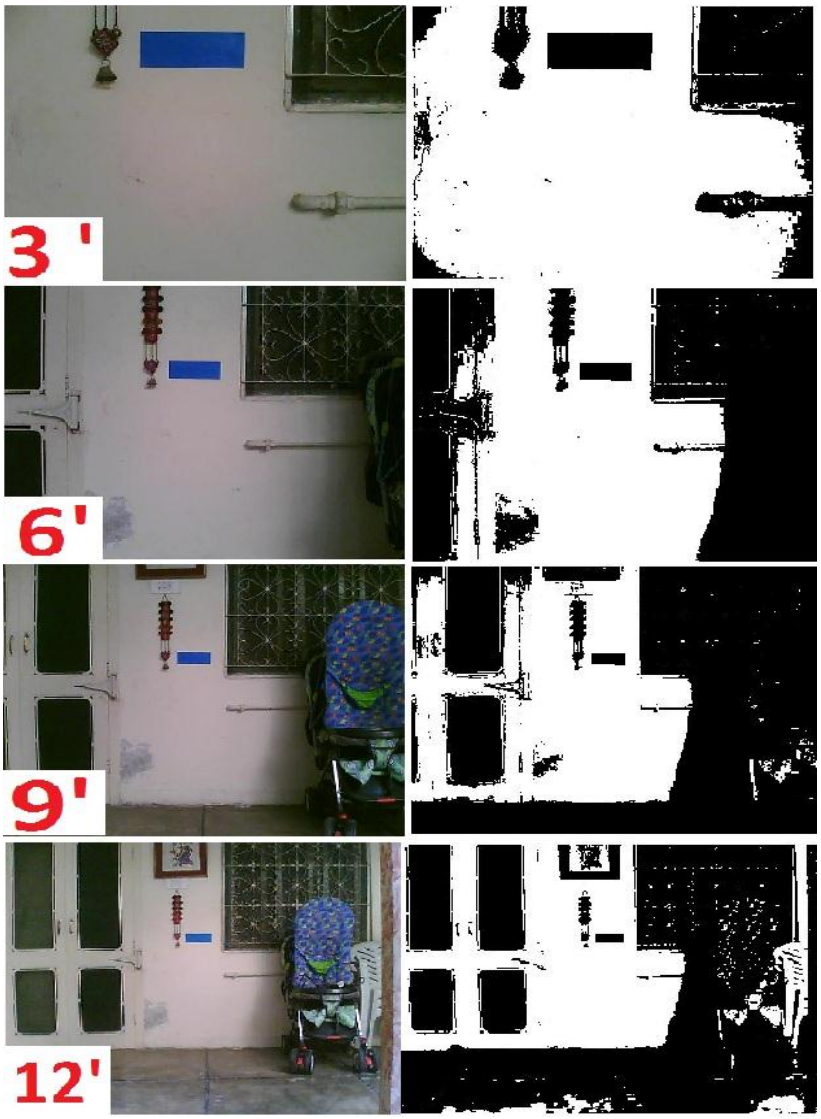

Figure 2.4: Sample data set (Images at 3, 6, 9, 12 Feet and their Binary Counterpart)

CHAPTER 3: IMAGE PROCESSING

Image processing is one of the major contributing steps in the algorithm. Sets of images gathered in the data collection phase will be required for distance calculations. At this stage, fresh snapshot from the input video feed is to be taken. All the captured images must undergo certain criteria before distance can be analyzed through them, any degradation in the image due to noise can be improved [9]. Image acquisition process will remain the same.

Processing [10] [11] is purely based on input images and technique used for that improvement. There is a separate method for refining respective degradation. All the images will not undergo through every step of image processing. It depends upon situation in hand that what method to be applied when. Designed algorithm should be capable enough that it should be applied selectively. Besides just improving the quality of the image, image is to be used for the distance estimation. Designed algorithm consists of some general set of protocols, by which not only necessary image processing is achieved but is also helpful for further processing. Let us consider figure 3.1 as an example.

Figure 3.1: (a) Acquire image (b) binary image of figure 3.1 (a)

.

Figure 3.1 contains a sample acquired image, and its binary appearance [12]. As it can be seen in the figure that image further consists of many other objects in the field of view. Our region of interest is the blue marker as shown in the figure. Corresponding to the blue colored marker shown in figure 3.1 (a) a clearly visible blob is displayed in the figure 3.1 (b).

3.1 Characteristics of Blobs

In order to compute the characteristics of every blob independently each blob is extracted individually [13]; therefore, image is converted into its logical 'NOT' as shown in figure 3.2

Figure 3.2: Logical NOT of input image

3.1.1 Logical NOT

Syntax used is:

```
% Logical NOT of Input Image
Ibw = ~im2bw(img);
```

As size and shape of maker is predefined, so is its characteristics except color. Figure 3.2 contains four independent objects displayed as white blobs. Each blob has certain amount of

connected white pixels. Noise if any is also reflected in the image, which entails a few connected pixels. A simple logical check can eliminate any such type of undesired noise. Single one line command can be used for the said task.

3.1.2 Reduction of Undesired Noise

```
% To Avoid Undesired Noise
Ibw2=bwareaopen(Ibw,100);
```

Any blob having undesired number of connected pixels can be neglected using this command [14]; selection of respective threshold value depends upon the resolution at which images were taken. By seeing the figure 3.2 one can analyze that there are only four blobs in the image, but how come the algorithm will automatically address the number of blobs in the image. Solution to the said problem is extraction of connected components [15] [16].

3.1.3 Extraction of Connected Components

```
% To Label and Count the Number of Connected Components
[L,Num]=bwlabel(Ibw,8);
```

Figure 3.3: Colored image and respective blobs shown independently

Further to this some inherent characteristics of every blob is to be calculated in order to select region of interest. Following properties are deliberated for every blob in the image.

3.1.4 Major Axis Length

Major axis length of every blob in the image.

```
% Loop from 1 to Maximum Number of Blobs in the Input Image
for i=1:Num
```

```
% Selection of ith Blob in the Image
sel=bwselect(L,i);

%MeasuringMajor Axis Length of all the Blobs in the Image
cal=regionprops(sel,'majoraxislength','minoraxislength','orientation');
maj_axis_len(i,1)=cat(1,cal.MajorAxisLength);

%Loop Termination
end
```

3.1.5 Minor Axis Length

Minor axis length of every blob in the image.

```
%Measuring Minor Axis Length of all the Blobs in the Image
cal=regionprops(sel,'majoraxislength','minoraxislength','orientation');
min_axis_len(i,1)=cat(1,cal.MinorAxisLength);
```

3.1.6 Orientation

Orientation of the blob with respect to the horizontal axis (where $\boldsymbol{\partial_x = 0}$)

```
%Measuring Orientation of all the Blobs in the Image
cal=regionprops(sel,'majoraxislength','minoraxislength','orientation');
orient(i,1)=cat(1,cal.Orientation);
```

3.1.7 Aspect Ratio

Aspect ratio of all the blobs in the image.

```
%Measuring Aspect Ratio of all the Blobs in the Image
cal=regionprops(sel,'majoraxislength','minoraxislength','orientation');
```

```
Aspect_Ratio(i,1)=(maj_axis_len(i,1))/(min_axis_len(i,1));
```

3.1.8 Area

All these computation were done with in a loop. Addition to this size and total covered area of the image is also calculated using:

```
% To Compute the Size of the Whole Image
[d1,d2]=size(Ibw);

% To Determine the Covered Area of the Image
Total_Area =(d1*d2);
```

3.1.9 Line Filters

In order to detect any vertical, horizontal or diagonal line segment in the image, certain filters were also designed:

- Horizontal

```
%Filter designed for Horizontal Lines
H=[-1 -1 -1; 2 2 2;-1 -1 -1];
e_rtd_img=edge(rtd_img);
DH_H=imfilter(e_rtd_img,H);
```

- Vertical

```
%Filter designed for Vertical Lines
V=[-1 2 -1;-1 2 -1;-1 2 -1];
```

- Diagonal

```
%Filters designed for Diagonal Lines
P45=[-1 -1 2;-1 2 -1;2 -1 -1];
M45=[2 -1 -1;-1 2 -1;-1 -1 2];
```

We will discuss filtering of some desired component in detail in the sections to come.

CHAPTER 4: DISTANCE CALCULATION

As the size and shape of the tag is fixed and is pre-defined, so is its properties [17]. Covered area by the tag in the image has certain proportion with that of overall covered area by the image. One can easily compute the actual area of the tag physically. Beside just covered area parameters that can be judged tangibly are:

- Length of major and minor axis of the tag
- Aspect ratio of the tag
- Color of the tag
- Number of corners of the tag
- Number of horizontal and vertical boundary lines (as the shape of the tag is rectangle so it must have two horizontal and two vertical boundary lines).

After analyzing the constraints of the tag, simple mathematical calculation can be used to obtain the covered area by the tag in the image. It can be referred as the actual area computed physically. Covered area of the tag in the image plays a vital role. It is used as an initial step in the designed algorithm to select the region of interest (tag) in the image. Examining the characteristics of the tag physically is only a one time measure and is to be considered as a part of camera calibration process. Once we have obtained both the physical (actual) as well as experimental properties of the tag, next step is the selection of the region of interest (ROI) in the image.

Experimental results of every blob in the image as shown in the figure 3.3 is compared with that of actual values of the tag respectively; blob which is nearly identical to the actual bounds is marked as ROI in the image. Comparing every blob at run time may cause delay in real environmental causing a lag in the performance. As already elaborated covered area by the tag in the image is important. It can be used to filter different blobs in the image. Any blob to be considered as a ROI must have covered area within certain range. Besides just comparing each value of every blob with the actual one, one can simply apply a threshold. Covered area by the tag can be used as a threshold. Blobs in the image having unexpected covered area as compared

to that of actual one can simply be neglected; in addition to this a simple check on aspect ratio of the remaining blobs in the image can effectively increase the performance of the algorithm.

Both covered area and the aspect ratio of the tag calculated in the phase - I is compared with the actual outcomes. After scrutinizing every blob in the image on the bases of only these two parameters, limited number of left over blobs in the image now have a greater chance of being selected as ROI. Comparing the experimental results of the remaining blobs in the image is now cost effective and is more efficient.

Finally, the blob having the least difference as compared to the actual dimensions of the tag is designated as ROI (with this object classification is done as shown in figure 4.1).

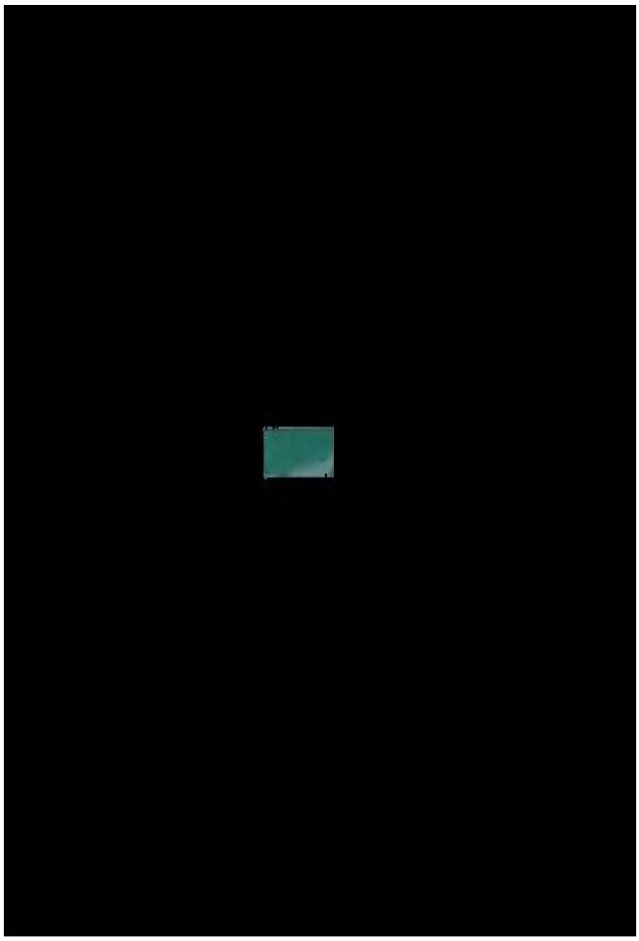

Figure 4.1: Object (ROI) Classification

As camera used for image acquisition in not mounted, it is not fixed. ROI in the image might have non zero orientation with respect to the horizontal axis. In order to make the

orientation of the tag zero, image is rotated as given below:

```
%Image is rotated to make the orientation zero
rtd_img=imrotate(Ibw,-orien);
```

Respective colored image of figure 4.1 at zero orientation is given in figure 4.2

Figure 4.2: ROI in the image at zero orientation

ROI is correctly identified using designed algorithm, it is completely visible in the figure 4.2 and is highlighted with blue boundary [18]. At this stage algorithm should have both covered area by the tag and correct identification of the tag in the image.

Next step is to evaluate the depth; once again covered area by the tag will be used to find the depth. Sequence of images gathered in the data collection phase with respect to the distance 'd' from the object will be used to drive a generalized mathematical expression. This

mathematical expression will produce the depth from a certain object in the work space, while considering covered area by the tag in the image as an input.

4.1 Distance Calculation

Before distance can be evaluated correctly algorithm requires some preprocessing or camera calibration. Images taken from different distances are analyzed. Actual distance ‘d’ from which the image is taken is known in the camera calibration process, corresponding covered area by the tag in respective image is also known. Covered area by the tag in the image is inversely proportional to the distance from which the image is taken. It is a natural phenomenon, objects near to the vision appears larger in size as compared to the one far away.

Finally, we have array of values containing distance ‘d’ from which the image is taken and respective covered area by the tag in the image as shown in the table below.

Table 4-1: Covered Area by the tag versus distance from which the image is taken

Ser	Covered Area	Distance ‘d’ (feet)
1	34264.375	2
2	17758.500	3
3	11183.125	4
4	7331.0000	5
5	5557.3750	6
6	3951.6250	7
7	3153.5000	8
8	2480.8750	9
9	1973.7500	10

As observed from the values illustrated in the table 4-1 covered area by the tag at two feet is more as compared to the others and as the distance increases covered area decreases. Values shown against the column covered area in the table 4-1 is the arithmetical mean of at least ten observations.

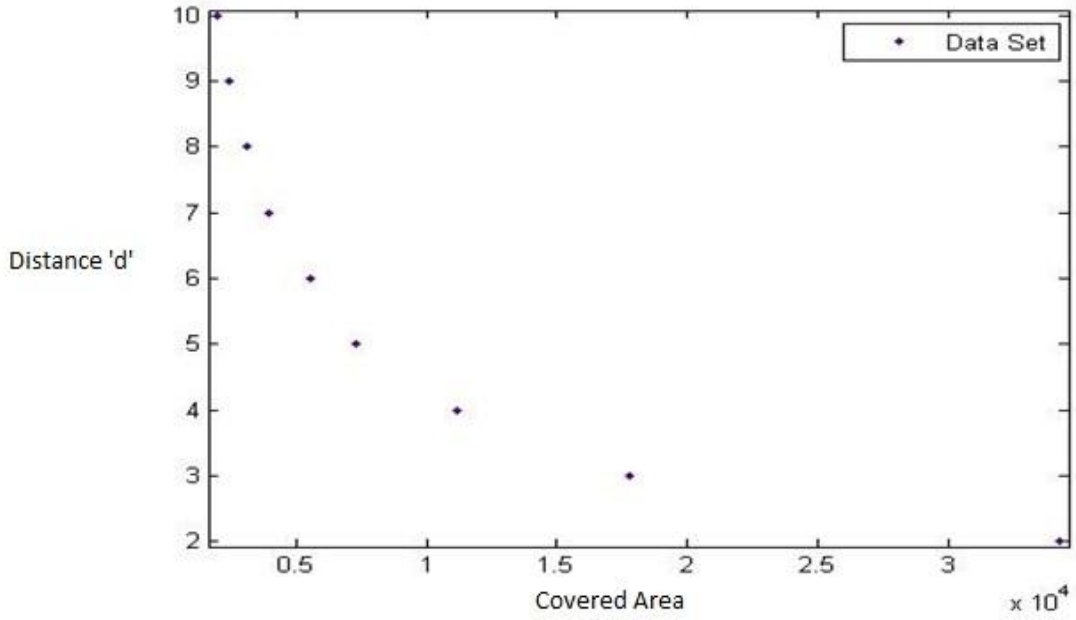

Figure 4.3: Plot of dataset versus covered area by the tag in the 2-D image

Figure 4.3 contains a data set of covered area by the tag in the 2-D image and respective distance from where the image is taken. In order to get a mathematical model which accurately demonstrates the pattern as shown in the figure 4.3, different curve fittings are to be applied. Each has its pros and cons but the one with least percentage error is selected.

4.2 Curve Fitting

In this section we will discuss some of the curve fitting and their characteristics [19] [20].

4.2.1 Quadratic Polynomial

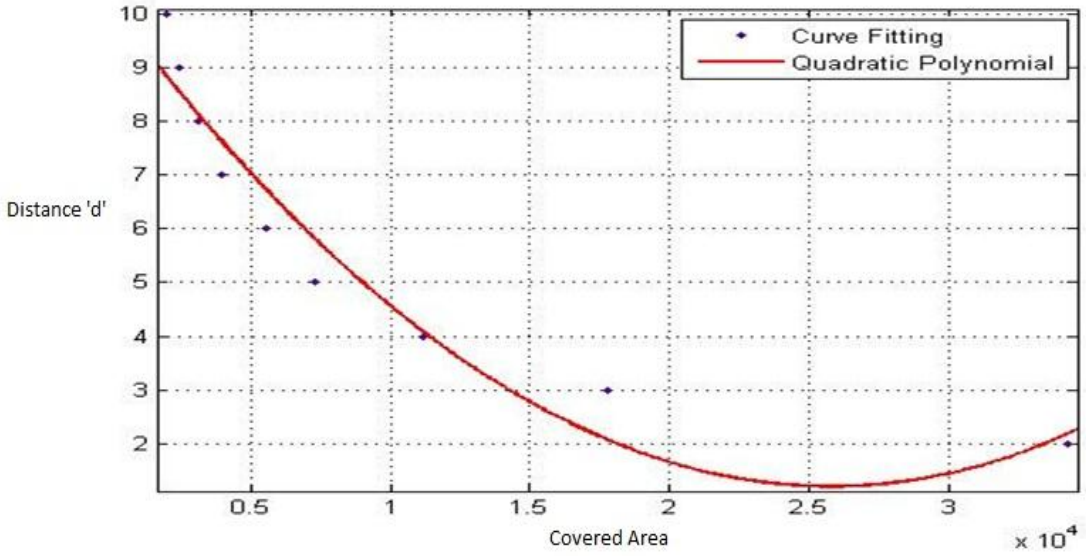

Figure 4.4: Curve fitting (Quadratic Polynomial)

$$\boldsymbol{f(x) = p1 * x^2 + p2 * x + p3} \tag{4.1}$$

Where;

$$\boldsymbol{p1 = 1.3513 - 008 \ (6.041e - 009, 2.099e - 008)}$$
$$\boldsymbol{p2 = \ -0.0006964 \ (-0.000968, -0.0004248)}$$
$$\boldsymbol{p3 = \ 10.18 \ (8.717, 11.64)}$$
$$\boldsymbol{x = \ Coveredareabythetag}$$
$$\boldsymbol{f(x) = \ Calculateddistancefromtheobject}$$

As seen in figure 4.4 results obtained using quadratic polynomial are not as desired.

4.2.2 Rational Polynomial

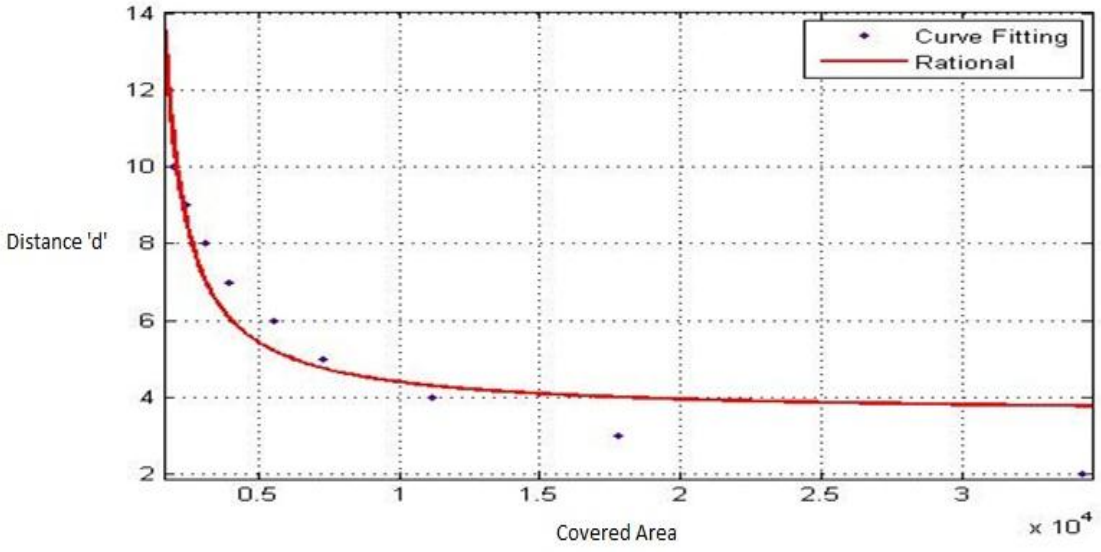

Figure 4.5: Curve fitting (Rational)

$$f(x) = \frac{p1 * x^2 + p2 * x + p3}{x^2 + q1 * x + q2} \qquad (4.2)$$

Similarly;

$$p1 = 3.518(-2.799, 9.835)$$
$$p2 = 4519(-1.309e + 011, 1.309e + 011)$$
$$p3 = 0.8415(-1.682e + 014, 1.682e + 014)$$
$$q1 = -918.6(-3.721e + 010, 3.721e + 010)$$
$$q2 = 15.32(-3.418e + 013, 3.418e + 013)$$
$$x = Coveredareabythetag$$
$$f(x) = Calculateddistancefromtheobject$$

Although, rational curve fitting is better as compared to the quadratic polynomial but power fit can be used to obtain more adequate results.

4.2.3 Power Fit

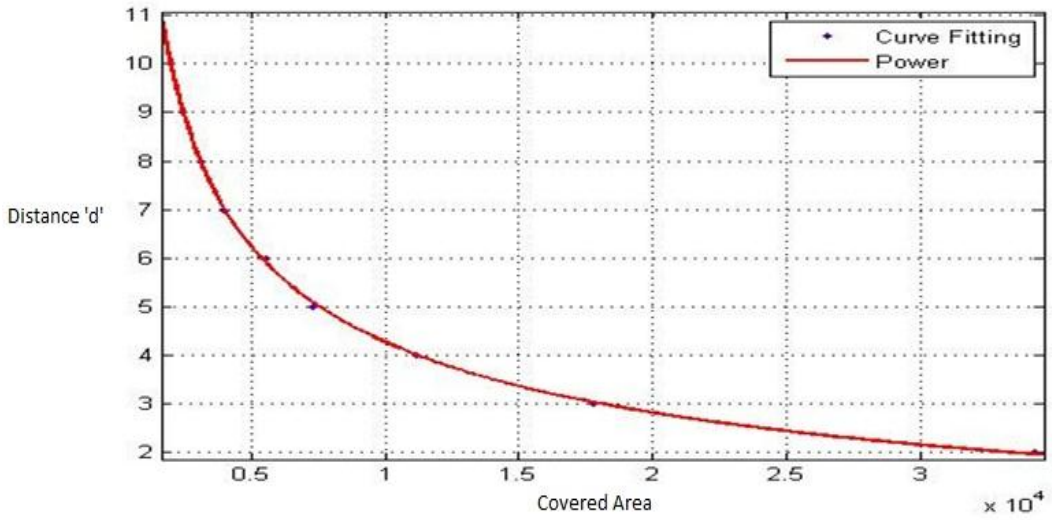

Figure 4.6: Curve fitting (Power Fit)

$$f(x) = a * x^b + c \qquad (4.3)$$

Where;

$$a = 327.8(203.8, 451.8)$$
$$b = -0.4444(-0.5031, -0.3857)$$
$$c = 10.18(8.717, 11.64)$$
$$x = Coveredareabythetag$$
$$f(x) = Calculateddistancefromtheobject$$

Results obtained using power fit is meaningful; thus equation 4.3 is considered as a final model equation which will be used for depth estimation. Where a, b and c are constants,$'x'$ represents covered area by a tag and function $'f(x)'$ is corresponding distance from which the image is taken. Algorithm uses this equation as a baseline for distance evaluation. There are different other constraints which take part in the selection criteria for best curve fitting equation. Table 4-2 illustrates the comparison of some curve fittings.

Table 4-2: Comparison of different curve fittings

Comparison				
Fit	**SSE**	**R-Square**	**Adjusted R-Square**	**RMSE**
Poly2	4.00900	0.9332	0.9109	0.8174
Poly5	0.03626	0.9994	0.9984	0.1099
Rational	7.72300	0.8713	0.7426	1.3900
Power2	0.03079	0.9995	0.9993	0.07163

In table 4-2 SSE corresponds to sum squared error, where R-Square is coefficient of determination (proportion of variation in the dependent variable explained by independent variable), RMSE is the root mean square error. Directional angle of the camera with respect to the tag is also important. Outcomes shown in table 4-3 have certain directional angle range, which lays between $90^{o} \pm 45^{o}$.

Now coming back to the point where the algorithm has selected the region of interest in the image after the image processing phase. At this stage algorithm have both the identified tag and covered area by the tag. Covered area by the tag can be used in equation 4.3 for depth estimation. Results obtained using power fit is depicted in table 4-3 and 4-4.

Table 4-3: Results (Power Fit)

Results (Power Fit)				
Distance (cm)	**TA**	**CA (E)**	**PE**	**CD (cm)**
106.68	307200	13950	1.48	107.204
106.68	307200	13431	2.32	110.337
167.64	307200	6342	2.3	169.564
167.64	307200	6082	1.82	168.102
304.80	307200	1969	0.24	304.932
304.80	307200	1982	0.41	304.992

In table 4-3 TA is total area covered by the image, where CA (E) is the calculated area covered by the tag using the designed algorithm, PE corresponds to percentage error, CD is the calculated distance from the object from where the image was taken and TP is the throughput of the algorithm. Distances are mentioned in centimeters.

.**Table 4-4:** Synopsis (Power Fit)

Synopsis			
Distance (cm)	**PE**	**CD (cm)**	**TP (sec)**
106.68	1.48	107.204	4.372
167.64	2.3	110.337	4.402
304.80	0.24	304.992	4.312

CHAPTER 5: POSITION LOCALIZATION AND GUI DEVELOPMENT

Set of values obtained using any algorithm should always form a gaussian curve. Repeated calculation should be performed in respective to check the biasness of the algorithm. Any designed algorithm should not produce influenced results. Similarly, biasness of the designed algorithm for depth evaluation was also checked, results are shown in table 5-1

Table 5-1: Experimental results (at zero orientation)

Calculated Distance			
Experiment	**5 (Feet)**	**6 (Feet)**	**7 (Feet)**
1	5.0303	6.0247	7.0492
2	5.0873	6.0077	7.0731
3	4.9778	5.9831	6.9507
4	5.0812	6.0567	7.0786
5	5.071	6.0544	7.0057
6	5.0482	5.9576	7.0003
7	5.0723	6.0111	7.0473
8	5.0948	6.01284	6.9542
9	5.0569	6.0361	7.0309
10	5.0152	5.9831	7.0565
11	5.0555	6.0201	6.9931
12	4.9893	6.0338	7.062
13	5.0329	5.9931	7.0021
14	5.0482	5.9998	6.9931
15	5.0488	6.0043	7.0021

5.1 Position Localization

Scope of the designed algorithm is limited to the indoor environment only. Dynamic position localization [21] has certain importance in any autonomous designed algorithm. Algorithm should be capable of identifying its coordinates at run time in order to update its position in the real time environment [22].Algorithm uses simple approach for the said task. Position is localized using depth estimation from any two adjacent walls within the closed environment. Point of intersection of two lines obtained in the depth estimation process will be the desired position. Position of the object will be updated dynamically using the estimated depth at every point.

Let us consider figure 5.1 as an example of any closed workspace for better understanding.

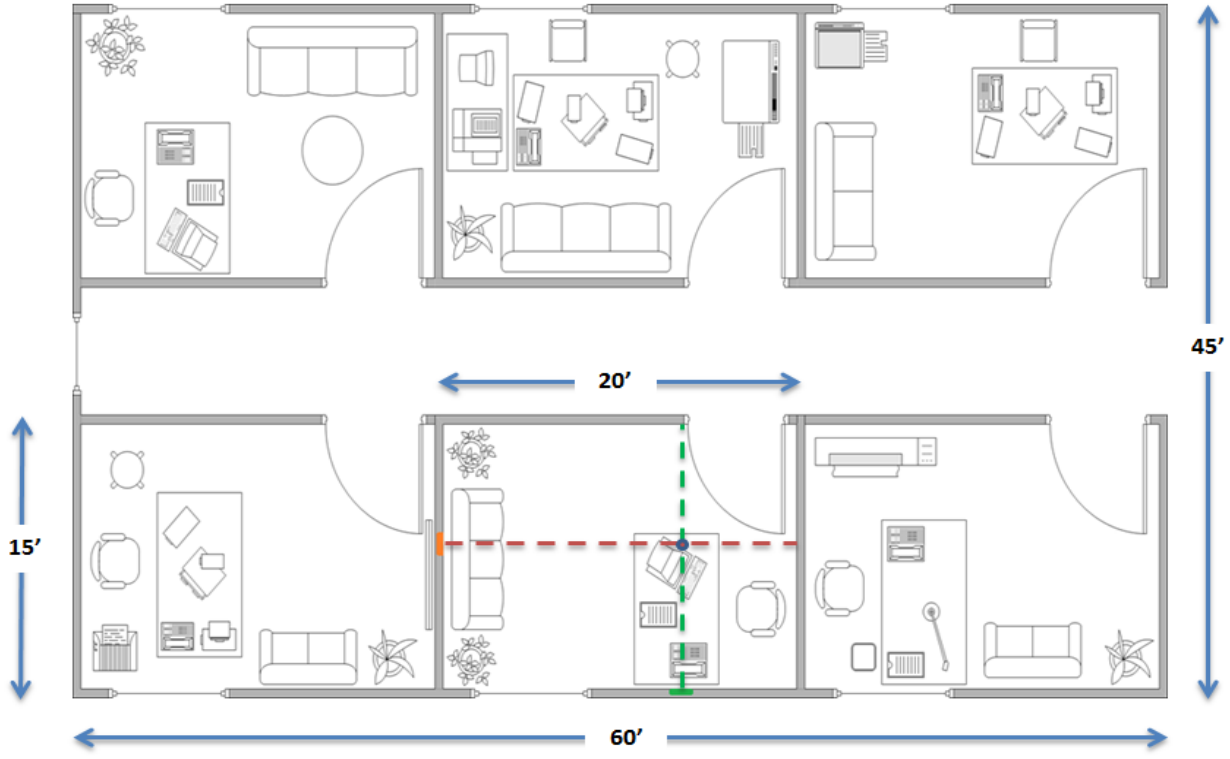

Figure 5.1: Closed workspace

As shown in the figure 5.1 having indoor environment we have fixed dimension. Workspace consists of 60' x 45' and the aspect ratio of the displayed rooms is 20' x 15'. To

localize the position algorithm first takes the images of the wall having red tag (ROI) and compute the distance from this respective wall, similarly for the green tagged wall.

Let us suppose the calculated distance from the red tag is 15 feet and 9 feet from the green tag. As the dimensions of the room are also known therefore, simple mathematical calculations can be used to determine the desired coordinates.

$$Length = 20' + 15' = 35' \text{ (Horizontal)}$$
$$Width = 0 + 9 = 9'(Vertical)$$

Respective global coordinates for the example shown in the figure 5.1 are (35, 9). As the research is limited to the pre-defined tag but as the research expands the algorithm will use every day routine objects as tag and will perform corresponding calculations.

5.2 GUI Development

Graphical User Interface (GUI) is purely designed in the matlab [23]. One can easily access the GUI environment in the matlab. Guide initiates the GUI design environment tools that allow GUIs to be created or edited interactively.

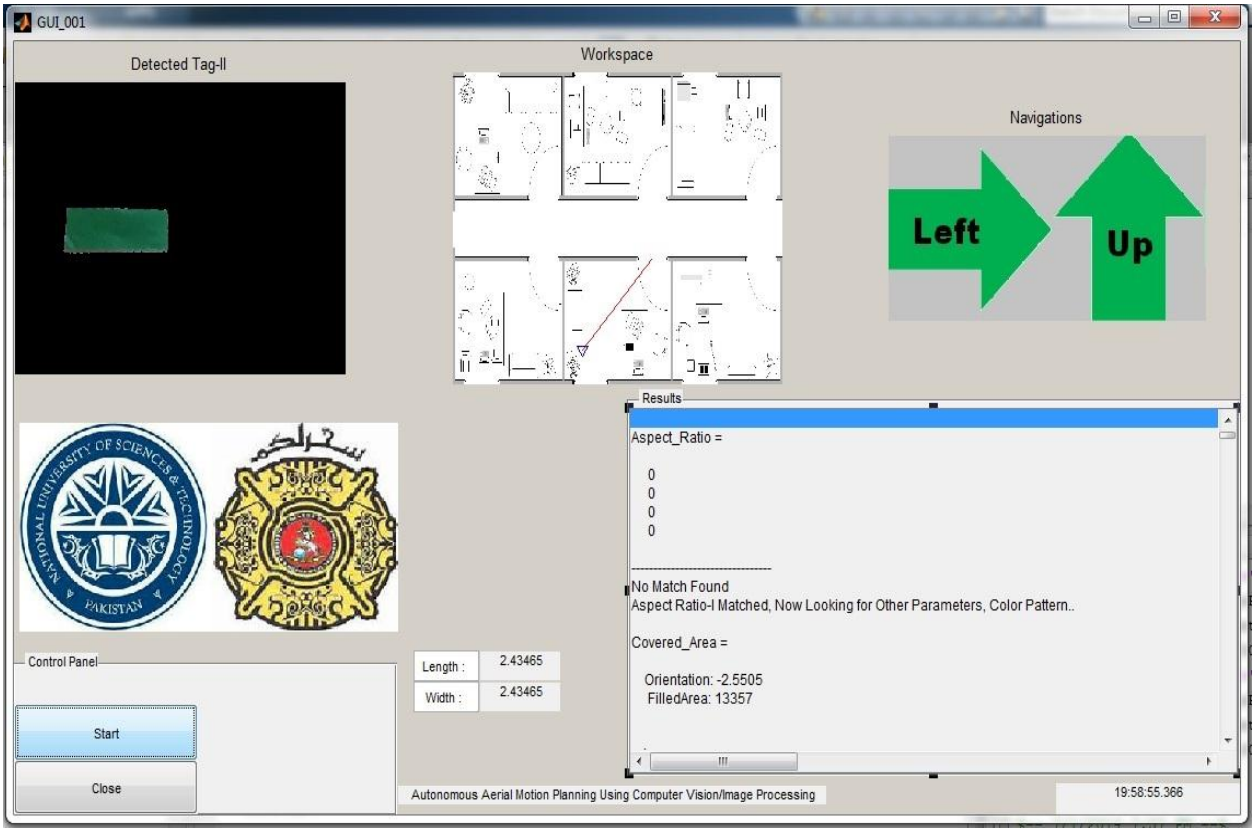

Figure 5.2: Graphical User Interface

Designed GUI as illustrated in figure 6.1 is divided into six major portions, identification of tag is displayed in the upper left corner. Workspace window demonstrates the map and position localization, diagonal line spreading away from the localized position is indicating the way out of the room. Upper right corner is the navigation for maneuvering out of the room; then we have estimated distances from the walls as length and width. Finally, we have results and control panel windows.

CHAPTER 6: CONCLUSION

Feature detection and extraction are well known techniques in computer vision and image processing. Already many other algorithms have been developed in this field. Most of these algorithms are limited to certain frame of reference. Such as corner points, lines, binary features, boundary traces extraction or detection. Likewise, derived algorithm has its own novelty element. Algorithm independently not only classifies a predefined tag from a single 2D image but also calculates distance to that object with accuracy. Secondly the algorithm works without any expensive equipment or additional aid such as sensors feedback which the predecessors do.

During the experiments performed so far observed throughput of this algorithm is 4.322 sec, whereas the maximum error is 8.16% which can be reduced by improving image acquisition and image processing process, so one can have high quality initial data.

The research shows that single camera depth estimation can be achieved using polynomial curve fitting approach. For a said marker of known dimensions one can determine a fixed equation which can then be used to find any random distance. The approach is efficient and can effectively be applied to applications of indoor navigation or motion planning algorithms. It is also not necessary to obtain expensive equipment for the said purpose.

6.1 Future Work

To make the algorithm intelligent enough that it can detect everyday objects as a marker instead of pre-defined shape and size of marker. Secondly, algorithm should be capable of classifying dynamic obstacles.

REFERENCES

[1] Real Time Motion Planning for Agile Autonomous Vechicles, Emilio Frazzoli, Munther A. Dahleh and Eric Feron, Journal of Guidance, Control and Dynamics. Vol. 25, No. 1, Jan - Feb 2002

[2] Learning depth from Single Monocular Images, Ashutosh Saxena, Sung H. Chung and Andrew Y. Ng, Stanford University Computer Science Department, Stanford CA94305

[3] Distance Measuring based on Stereoscopic Pictures, Jernej Mrovlje and Damir Vrancic. 9th PhD Workshop on Systems and Control, Young Generation Viewpoint, 1-3 Oct 2008, Izola, Slovenia

[4] E.R Davies, Law's Texture Energy in TEXTURE. In Machine Vision: Theory, Algorithms, Practicalities, 2nd Edition, Academic Press, San Diego.

[5] 3-D Depth Reconstruction from a Single Still Image, Ashutosh Saxena, Sung H. Chung, Andrew Y. NG, Computer Science Department, Stanford University, Stanford

[6] Undergraduate Physics Students' Computing And Learning Environment (Physics Virtual Bookshelf), David M. Harrison, Department of Physics, University of Toronto

[7] Real time Moving Obstacle Detection using Optical Flow Models, Christophe Braillon, Cedric Pradalier, James L. Crowley, and Christian Laugier, Intelligent Vehicles Symposium 2006, June 13-15-2006, Tokyo Japan

[8] Depth Estimation from a Single Camera Image using Power Fit, Muhammad Umair Akhlaq, Umar Shahbaz, Umer Izhar, International Conference on Robotics and Emerging Allied Technologies in Engineering (iCREATE), 2014

[9] Image Processing – Principles and Applications, Tinku Acharya, Ajoy K.Ray, A Wiley-Interscience Publication

[10] The Image Processing Handbook, 2nd Edition, Russ, Jhon C.; Woods, Roger P.M.D

[11] Introductory Digital Image Processing: A Remote Sensing Perspective, Jensen, J.R., ISBN 0-13-205840-5

[12] MATLAB vs Intellect: Image Processing for Robot Obstacle Avoidance, James Painter, Department of Physics & Engineering, Elizabethtown College

[13] Speed Up Robust Features (SURF) , Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Elsevier- 5 September 2007

[14] Noise Reduction in the Digital Images, Lana J. Jobes, Carlson Center for Imaging Science, Rochester Institute of Technology, May 1999

[15] Automatic Text Location in the Images and Video Frames, Anil K. Jain, Bin YU, Department of Computer Science, Michigan State University, East Lansing, MI 48824-1027,USA

[16] Automatic Image Segmentation by Integrating Color-Edge Extraction and Seeding Region Growing, Jianping Fan, Department of Computer Science, Purdue University, West Lafayette, IN, USA, Yau, D.K.Y. ; Elmagarmid, A.K. ; Aref, W.G., 7 August 2002

[17] Image Processing, Analysis and Machine Vision, 4th Edition, Milan Sonka, Vaclav Hlavac and Roger Boyle

[18] Contour Detection and Hierarchical Image Segmentation, Arbelaez P, Department of Electrical Engineering & Computer Science, University of California at Berkeley, CA, USA, Maire M, Fowlkes C, Malik J, 26 August 2010

[19] Numerical Methods of Curve Fitting, P.G. Guest, Philip George Guest, Cambridge University Press

[20] The Comparison Research of Non Linear Curve Fitting in Matlab and LabVIEW, Hao Wen, Beijing Changcheng Aeronaut. Meas. & Control Technol. Res. Inst., Aviation Ltd. Corp of China, Beijing, China, Jing Ma, Meiju Zhang, Guimei Ma, 24 – 27 June 2012

[21] Robust Vehicle Localization in Urban Environments using Probabilistic Maps, Levinson J, Stanford Artificial Intell. Lab, Stanford, CA, USA, Thrun S, 3-7 May 2010

[22] Humanoid Robot Localization in Complex Indoor Environments, Hornung A, Department of Computer Science, University of Freiburg, Germany, Wurm K M, Bennewitz M, 18-22 Oct 2010

[23] Learning to Program with Matlab: Building GUI Tools, Craig S. Lent

APPENDIX A

Syntax used for algorithm and GUI development is illustrated below:

```
function varargout = GUI_001(varargin)
% GUI_001 MATLAB code for GUI_001.fig
%      GUI_001, by itself, creates a new GUI_001 or raises the existing
%      singleton*.
%
%      H = GUI_001 returns the handle to a new GUI_001 or the handle to
%      the existing singleton*.
%
%      GUI_001('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_001.M with the given input arguments.
%
%      GUI_001('Property','Value',...) creates a new GUI_001 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before GUI_001_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to GUI_001_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUI_001

% Last Modified by GUIDE v2.5 23-Jun-2014 00:46:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @GUI_001_OpeningFcn, ...
'gui_OutputFcn',  @GUI_001_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before GUI_001 is made visible.
function GUI_001_OpeningFcn(hObject, eventdata, handles, varargin)
```

```
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to GUI_001 (see VARARGIN)

% Choose default command line output for GUI_001
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI_001 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = GUI_001_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
diary('test.txt')
diary on
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Create a video input object.

vid = videoinput('winvideo',1,'YUY2_640x480');

% Create a figure window. This example turns off the default
% toolbar and menubar in the figure.
%hFig = figure('Toolbar','none',...
%      'Menubar', 'none',...
%     'NumberTitle','Off',...
%    'Name','Prototype: (Autonomous Aerial Motion Planning)');

% Set up the push buttons
%uicontrol('String', 'Preview On',...
%   'Callback', 'preview(vid)',...
%  'Units','normalized',...
% 'Position',[0 0 0.15 .07]);
%uicontrol('String', 'Preview Off',...
%   'Callback', 'stoppreview(vid)',...
%  'Units','normalized',...
% 'Position',[.17 0 .15 .07]);
```

```
uicontrol('String', 'Close',...
'Callback', 'close(gcf)',...
'Units','normalized',...
'Position',[0 0 .15 .07]);     % [0 0 .15 .07]

% Create the text label for the timestamp
hTextLabel = uicontrol('style','text','String','Timestamp', ...
'Units','normalized',...
'Position',[0.85 -.04 .15 .08]);

% Create the image object in which you want to
% display the video preview data.
vidRes = get(vid, 'VideoResolution');
imWidth = vidRes(1);
imHeight = vidRes(2);
nBands = get(vid, 'NumberOfBands');
hImage = image( zeros(imHeight, imWidth, nBands) );

% Specify the size of the axes that contains the image object
% so that it displays the image at the right resolution and
% centers it in the figure window.
figSize = get(handles.axes3,'Position');
figWidth = figSize(3);
figHeight = figSize(4);
set(gca,'unit','pixels',...
'position',[800 345 290 255]);%[ ((figWidth - imWidth)/2)... [800 298 290
255]
% ((figHeight - imHeight)/2)...
% imWidth imHeight ]);

% Set up the update preview window function.
setappdata(hImage,'UpdatePreviewWindowFcn',@mypreview_fcn);

% Make handle to text label available to update function.
setappdata(hImage,'HandleToTimestampLabel',hTextLabel);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% INITIALIZATION/BASIC CALC %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

preview(vid, hImage);
axes(handles.axes1)
match=0;
%IP_1=inputdlg('Length :','Workspace Dimensions',[1 50]);
%Len = str2num(IP_1{:});
%IP_2=inputdlg('Width :','Workspace Dimensions',[1 50]);
%Wid = str2num(IP_2{:});
Len = 15;
Wid = 10;
ar_mth=0;
tic;
%set(vid,'ReturnedColorSpace','RGB');
while (ar_mth~=1)
set(vid,'ReturnedColorSpace','RGB');
% For Video Input
```

```
%start(vid);    % For Video Input
pause (5)
img=getsnapshot(vid);
% For Video Input
%img=imread('rb25_10f.jpg');
% Neglect Comment for Single Image
imgs=img;
Ibw = ~im2bw(img);
% Graythresh(img));
Ibw2=bwareaopen(Ibw,100);
% In case to avoid noise.
Ibw=Ibw2;
[L,Num]=bwlabel(Ibw,8);
[d1,d2]=size(Ibw);
Total_Area =(d1*d2);
% disp('Total Area in Terms of Pixels') % Total Area (In Pixels)
maj_axis_len=zeros(Num,1);
min_axis_len=zeros(Num,1);
orient=zeros(Num,1);
Aspect_Ratio=zeros(Num,1)
mn_a_r=2.65;
mx_a_r=2.85;
disp('---------------------------------')
%figure(10);
imshow(img); title('Camera Image-I'),pause (0.5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% CALC OF INTRINSIC PARAMETERS %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:Num
    sel=bwselect(L,i);
    cal=regionprops(sel,'majoraxislength','minoraxislength','orientation');
    maj_axis_len(i,1)=cat(1,cal.MajorAxisLength);
    min_axis_len(i,1)=cat(1,cal.MinorAxisLength);
    orient(i,1)=cat(1,cal.Orientation);
    Aspect_Ratio(i,1)=(maj_axis_len(i,1))/(min_axis_len(i,1));
% Confirm Aspect Ratio.
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% FILERS FOR LINES DETECTION %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H=[-1 -1 -1; 2 2 2;-1 -1 -1];
% Horizontal Line Filter
V=[-1 2 -1;-1 2 -1;-1 2 -1];
% Vertical Line Filter
P45=[-1 -1 2;-1 2 -1;2 -1 -1];
% D_1@45 Line Filter
M45=[2 -1 -1;-1 2 -1;-1 -1 2];
% D_2@45 Line Filter
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% IDENTIFICATION OF TAG %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:Num
if (Aspect_Ratio(j,1)>=mn_a_r)&&(Aspect_Ratio(j,1)<=mx_a_r)
        ar_mth=1;
        disp('Aspect Ratio-I Matched, Now Looking for Other Parameters, Color
Pattern..');
        Ibw=bwselect(L,j);
%figure(15);
        imshow(Ibw); title('Camera Image (White & Black)-I'),pause (0.5)
        Covered_Area=regionprops(Ibw,'FilledArea','orientation')
% Size of Filled Area (In Pixels)
        orien=cat(1,Covered_Area.Orientation)
        rtd_img=imrotate(Ibw,-orien);
%figure(20);
        imshow(rtd_img); title('Zero Orientation Camera Image (White &
Black)-I'),pause (0.5)
        e_rtd_img=edge(rtd_img);
%  figure (25); imshow(e_rtd_img);
        DH_H=imfilter(e_rtd_img,H);
        DH_H=bwareaopen(DH_H,50);
%figure (25),
        imshow(e_rtd_img),title('Edge Detection-I'),pause (0.5)
%figure (26),
        imshow(DH_H),title('Horizontal Line Detection-I'),pause (0.5);
        C=corner(e_rtd_img);
% figure(27),
        imshow(e_rtd_img);%pause (0.75);
        hold on
        plot(min(C(:,1)), min(C(:,2)), 'r*');
        plot(max(C(:,1)), min(C(:,2)), 'r*');
        plot(min(C(:,1)), max(C(:,2)), 'r*');
        plot(max(C(:,1)), max(C(:,2)), 'r*');

        [La, Na_H]=bwlabel(DH_H);
        Na_H;
        DH_V=imfilter(e_rtd_img,V);
        DH_V=bwareaopen(DH_V,40);
%figure (28), imshow(e_rtd_img),title('Original Image')
%figure(28),
        imshow(DH_V),title('Vertical Line Detection-I'),pause (0.5);
        [La, Na_V]=bwlabel(DH_V);
        Na_V;


if (Na_H+Na_V==4)
            disp('Second Condition-I also Satisfied, Entering Next Level
..');
end

        Ifill = imfill(rtd_img,'holes');
        %Ifill = imfill(Ibw,'holes');
        Iarea = bwareaopen(Ifill,100);
        Ifinal = bwlabel(Iarea);
```

```
        [s1 s2]= size(Ifinal)
        stat = regionprops(Ifinal,'boundingbox');
% figure(30)
        rt_i=imrotate(img,-orien);
        imshow(rt_i),title('Tag Detection (At Zero Orientation)-I'); hold on;
for cnt = 1 : numel(stat)
                        bb = stat(cnt).BoundingBox;

rectangle('position',bb,'edgecolor','g','linewidth',2.5);
end
        match=1;

break
else
        ar_mth=0;
        disp('No Match Found');
        stop(vid);
%closepreview(vid);
%  closepreview
% close all
end



end
end
%stop (vid);
%closepreview (vid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% TAG SEGMENTATION FROM IMGAGE %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (match==1)
   cum=double(0);
   cnt=0;
for i=1:s1
for j=1:s2
if Ifinal(i,j)==1
                cnt=cnt+1;
                rt_i(i,j,1)=rt_i(i,j,1);
                rt_i(i,j,2)=rt_i(i,j,2);
                rt_i(i,j,3)=rt_i(i,j,3);
%a=cum;
%b=rt_i(i,j);
%c=double(rt_i(i,j));
                cum=(double(rt_i(i,j))+cum);
else
                rt_i(i,j,1)=0;
                rt_i(i,j,2)=0;
                rt_i(i,j,3)=0;
end
end
end
a_sum=(cum/cnt)
%figure(40)
imshow(rt_i),title('Detected Tag-I'),pause (0.5);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% COLOR SEGMENTATION (TAG) %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R=imgs(:,:,1);
G=imgs(:,:,2);
B=imgs(:,:,3);
R_M=max(max(R));
G_M=max(max(G));
B_M=max(max(B));
if (R_M>G_M)&&(R_M>B_M)
    disp('---------------------------------')
    disp('Tag-I is Red.')
end
if (G_M>R_M)&&(G_M>B_M)
    disp('---------------------------------')
    disp('Tag-I is Green.')
end
if (B_M>R_M)&&(B_M>G_M)
    disp('---------------------------------')
    disp('Tag-I is Blue.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CALCULATING AREA OF TAG %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Area_Cal=cat(1,Covered_Area.FilledArea)
a =       248;
b =     -0.469;
c =     -0.2109;
Estd_Dist=a*Area_Cal^b+c
end

%Area_Cal=cat(1,Covered_Area.FilledArea)
%a =       354.2;
%b =     -0.4944;
%c =     -0.5793;
%Estd_Dist=a*Area_Cal^b+c
%end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% SECOND PART GOES HERE %%%%%%%
%%%%% INITIALIZATION/BASIC CALC %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ar_mth_1=0;
%set(vid,'ReturnedColorSpace','RGB');
while (ar_mth_1~=1)
set(vid,'ReturnedColorSpace','RGB');
% For Video Input
start(vid);    % For Video Input
pause (5)
img_1=getsnapshot(vid);
% For Video Input
%img=imread('rb25_10f.jpg');
```

```
% Neglect Comment for Single Image
imgs_1=img_1;
Ibw_1 = ~im2bw(img_1);
% Graythresh(img));
Ibw2_1=bwareaopen(Ibw_1,100);
% In case to avoid noise.
Ibw_1=Ibw2_1;
[L_1,Num_1]=bwlabel(Ibw_1,8);
[d1_1,d2_1]=size(Ibw_1);
Total_Area_1 =(d1_1*d2_1);
% disp('Total Area in Terms of Pixels') % Total Area (In Pixels)
maj_axis_len_1=zeros(Num_1,1);
min_axis_len_1=zeros(Num_1,1);
orient_1=zeros(Num_1,1);
Aspect_Ratio_1=zeros(Num_1,1)
mn_a_r_1=2.65;
mx_a_r_1=2.85;
disp('--------------------------------')
%figure(10);
imshow(img_1); title('Camera Image-II'),pause (0.5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% CALC OF INTRINSIC PARAMETERS %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:Num_1
    sel_1=bwselect(L_1,i);

cal_1=regionprops(sel_1,'majoraxislength','minoraxislength','orientation');
    maj_axis_len_1(i,1)=cat(1,cal_1.MajorAxisLength);
    min_axis_len_1(i,1)=cat(1,cal_1.MinorAxisLength);
    orient_1(i,1)=cat(1,cal_1.Orientation);
    Aspect_Ratio_1(i,1)=(maj_axis_len_1(i,1))/(min_axis_len_1(i,1));
% Confirm Aspect Ratio.
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% FILERS FOR LINES DETECTION %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H_1=[-1 -1 -1; 2 2 2;-1 -1 -1];
% Horizontal Line Filter
V_1=[-1 2 -1;-1 2 -1;-1 2 -1];
% Vertical Line Filter
P45_1=[-1 -1 2;-1 2 -1;2 -1 -1];
% D_1@45 Line Filter
M45_1=[2 -1 -1;-1 2 -1;-1 -1 2];
% D_2@45 Line Filter

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% IDENTIFICATION OF TAG %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:Num_1
if (Aspect_Ratio_1(j,1)>=mn_a_r_1)&&(Aspect_Ratio_1(j,1)<=mx_a_r_1)
        ar_mth_1=1;
```

```
        disp('Aspect Ratio-II Matched, Now Looking for Other Parameters,
Color Pattern..');
        Ibw_1=bwselect(L_1,j);
%figure(15);
        imshow(Ibw_1); title('Camera Image (White & Black)-II'),pause (0.5)
        Covered_Area_1=regionprops(Ibw_1,'FilledArea','orientation')
        % Size of Filled Area (In Pixels)
        orien_1=cat(1,Covered_Area_1.Orientation);
        rtd_img_1=imrotate(Ibw_1,-orien_1);
%figure(20);
        imshow(rtd_img_1); title('Zero Orientation Camera Image (White &
Black)-II'),pause (0.5)
        e_rtd_img_1=edge(rtd_img_1);
% figure (25); imshow(e_rtd_img);
        DH_H_1=imfilter(e_rtd_img_1,H_1);
        DH_H_1=bwareaopen(DH_H_1,50);
%figure (25),
        imshow(e_rtd_img_1),title('Edge Detection-II'),pause (0.5)
%figure (26),
        imshow(DH_H_1),title('Horizontal Line Detection-II'),pause (0.5);
        C_1=corner(e_rtd_img_1);
% figure(27),
        imshow(e_rtd_img_1);%pause (0.75);
        hold on
        plot(min(C_1(:,1)), min(C_1(:,2)), 'r*');
        plot(max(C_1(:,1)), min(C_1(:,2)), 'r*');
        plot(min(C_1(:,1)), max(C_1(:,2)), 'r*');
        plot(max(C_1(:,1)), max(C_1(:,2)), 'r*');

        [La_1, Na_H_1]=bwlabel(DH_H_1);
        Na_H_1;
        DH_V_1=imfilter(e_rtd_img_1,V);
        DH_V_1=bwareaopen(DH_V_1,40);
%figure (28), imshow(e_rtd_img),title('Original Image')
%figure(28),
        imshow(DH_V_1),title('Vertical Line Detection-II'),pause (0.5);
        [La_1, Na_V_1]=bwlabel(DH_V_1);
        Na_V_1;


if (Na_H_1+Na_V_1==4)
            disp('Second Condition-II also Satisfied, Entering Next Level
..');
end

        Ifill_1 = imfill(rtd_img_1,'holes');
        % Ifill = imfill(Ibw,'holes');
        Iarea_1 = bwareaopen(Ifill_1,100);
        Ifinal_1 = bwlabel(Iarea_1);
        [s1_1 s2_1]= size(Ifinal_1)
        stat_1 = regionprops(Ifinal_1,'boundingbox');
% figure(30)
        rt_i_1=imrotate(img_1,-orien_1);
        imshow(rt_i_1),title('Tag Detection (At Zero Orientation)-II'); hold
on;
for cnt = 1 : numel(stat_1)
```

```
                        bb_1 = stat_1(cnt).BoundingBox;

rectangle('position',bb_1,'edgecolor','g','linewidth',2.5);
end
        match_1=1;

break
else
        ar_mth_1=0;
        disp('No Match Found');
        stop(vid);
% closepreview(vid);
% closepreview
% close all
end



end
end
%stop (vid);
%closepreview (vid);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% TAG SEGMENTATION FROM IMGAGE %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (match_1==1)
   cum_1=double(0);
   cnt_1=0;
for i=1:s1_1
for j=1:s2_1
if Ifinal_1(i,j)==1
                cnt_1=cnt_1+1;
                rt_i_1(i,j,1)=rt_i_1(i,j,1);
                rt_i_1(i,j,2)=rt_i_1(i,j,2);
                rt_i_1(i,j,3)=rt_i_1(i,j,3);
%a=cum;
%b=rt_i(i,j);
%c=double(rt_i(i,j));
                cum_1=(double(rt_i_1(i,j))+cum_1);
else
                rt_i_1(i,j,1)=0;
                rt_i_1(i,j,2)=0;
                rt_i_1(i,j,3)=0;
end
end
end
a_sum_1=(cum_1/cnt_1)
%figure(40)
imshow(rt_i_1),title('Detected Tag-II');%pause (0.75);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% COLOR SEGMENTATION (TAG) %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R_1=imgs_1(:,:,1);
G_1=imgs_1(:,:,2);
B_1=imgs_1(:,:,3);
R_M_1=max(max(R_1));
G_M_1=max(max(G_1));
B_M_1=max(max(B_1));
if (R_M_1>G_M_1)&&(R_M_1>B_M_1)
    disp('--------------------------------')
    disp('Tag is Red.')
end
if (G_M_1>R_M_1)&&(G_M_1>B_M_1)
    disp('--------------------------------')
    disp('Tag is Green.')
end
if (B_M_1>R_M_1)&&(B_M_1>G_M_1)
    disp('--------------------------------')
    disp('Tag is Blue.')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CALCULATING AREA OF TAG %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Area_Cal_1=cat(1,Covered_Area_1.FilledArea)
a_1 =       248;
b_1 =     -0.469;
c_1 =     -0.2109;
Estd_Dist_1=a_1*Area_Cal_1^b_1+c_1
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% POSITION LOCALIZATION  %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(Estd_Dist>Len||Estd_Dist_1>Wid);disp('Invalid Workspace Dimensions')
else
        dim_1=Estd_Dist;
        dim_2=Estd_Dist_1;
end

disp(['"X" Coordinate is: ', num2str(dim_1)])
disp(['"Y" Coordinate is: ', num2str(dim_2)])
disp('--------------------------------')
disp('--------------------------------')
axes(handles.axes2)
fimg=imread('WL-ID.jpg');
[fm fn]=size(fimg);
imagesc(fimg)
hold on;
t1=double(746 + (dim_1*45.733))
t2=double(1195 - (dim_2*46.33))
plot(t1,t2,'V');
```

```
%hold on;
plot([t1,1321],[t2 727],'r')
axis off;
title('Workspace');
xlabel('Length');
ylabel('Width');
Total_Time_Spent=toc

if (t1<1236)
axes(handles.axes3)
imshow('1.jpg')
title('Navigations')

pause (1)
axes(handles.axes3)
imshow('2.jpg')
title('Navigations')

pause (1)
axes(handles.axes3)
imshow('3.jpg')
title('Navigations')

pause (1)
axes(handles.axes3)
imshow('1.jpg')
title('Navigations')

pause (1)
axes(handles.axes3)
imshow('2.jpg')
title('Navigations')

pause (1)
axes(handles.axes3)
imshow('3.jpg')
title('Navigations')

else
axes(handles.axes3)
imshow('uu.jpg')
title('Navigations')
end
diary off
f = GUI_001('menu','none','toolbar','none');
    fid = fopen('test.txt');
    ph = uipanel(f,'Units','normalized','position',[0.5 0.05 0.5
0.5],'title',...
'Results');
    lbh = uicontrol(ph,'style','listbox','Units','normalized','position',...
        [0 0 1 1],'FontSize',9);

    indic = 1;
while 1
         tline = fgetl(fid);
```

```
if ~ischar(tline),
break
end
         strings{indic}=tline;
         indic = indic + 1;
end
    fclose(fid);
    set(lbh,'string',strings);
    set(lbh,'Value',1);
    set(lbh,'Selected','on');

set(handles.text4,'String',Estd_Dist);
set(handles.text6,'String',Estd_Dist);

% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```