SINGLE CAMERA-BASED VISION SYSTEMS FOR GROUND &
AERIAL ROBOTS

A PhD dissertation
Presented to
The Academic Faculty

by

Syed Irtiza Ali Shah

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
School of AEROSPACE ENGINEERING

Georgia Institute of Technology
December 2010
SINGLE CAMERA-BASED VISION SYSTEMS FOR GROUND & AERIAL ROBOTS

Approved by:

Dr. Eric N Johnson, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Daniel P Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Eric Feron
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Mark Costello
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Phillip Jones
Adaptive Flight Inc.
Atlanta Georgia

Date Approved: July 23rd, 2010
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr Eric Johnson, along with other committee members for evaluating this dissertation and for providing me with the requisite guidance and support, in order to successfully complete this work.

I also owe a lot of thanks to my family, friends and colleagues at Georgia Tech - specifically at UAVRF, whose help, support, well-wishes and prayers brought me here.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xiii</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>xiv</td>
</tr>
</tbody>
</table>

## CHAPTER

1 INTRODUCTION

- Overview
  - 1
- Summary of Contributions
  - 3
- Organization of Dissertation
  - 4

PART I: OBJECT DETECTION USING IMAGES FROM A SINGLE CAMERA FOR SEARCH AND IDENTIFICATION

2 INTRODUCTION TO SEARCH & IDENTIFICATION PROBLEM (PART I)

- Related Research
  - 8

3 PRELIMINARY EXPERIMENTS AND INVESTIGATION

- Grayscale Preliminary Analysis
  - 13
- Analysis of Color Spaces
  - 14
- Analysis Based on Camera Filters
  - 16

4 DETAILED ANALYSIS AND PROPOSED BLOCK BASED SEARCH ALGORITHMS

- 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed Analysis</td>
<td>18</td>
</tr>
<tr>
<td>Multiple Thresholding Algorithm and Testing</td>
<td>25</td>
</tr>
<tr>
<td>Linear Confidence Level Algorithm and Testing</td>
<td>28</td>
</tr>
<tr>
<td><strong>5 IMPLEMENTATION AND FLIGHT TESTING</strong></td>
<td>33</td>
</tr>
<tr>
<td>Implementation Platform &amp; Onboard Hardware</td>
<td>34</td>
</tr>
<tr>
<td>Onboard Software Overview</td>
<td>36</td>
</tr>
<tr>
<td>HITL tests and Flight Tests</td>
<td>38</td>
</tr>
<tr>
<td><strong>6 NOISE HANDLING AND FINAL ALGORITHM</strong></td>
<td>40</td>
</tr>
<tr>
<td>Noise Analysis and Handling</td>
<td>40</td>
</tr>
<tr>
<td>Final Implemented Algorithm</td>
<td>43</td>
</tr>
<tr>
<td><strong>7 CONCLUDING SEARCH &amp; IDENTIFICATION PROBLEM (PART I)</strong></td>
<td>46</td>
</tr>
<tr>
<td><strong>PART II: OBJECT DETECTION USING IMAGES FROM A SINGLE CAMERA FOR COLLISION AVOIDANCE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>8 OVERVIEW OF EXISTING APPROACHES FOR COLLISION AVOIDANCE</strong></td>
<td>48</td>
</tr>
<tr>
<td>Introduction</td>
<td>48</td>
</tr>
<tr>
<td>Multiple Sensor Based Obstacle Detection &amp; Avoidance</td>
<td>50</td>
</tr>
<tr>
<td>Single Sensor Based Approaches</td>
<td>51</td>
</tr>
<tr>
<td>Other Three Dimensional Approaches</td>
<td>52</td>
</tr>
<tr>
<td>Obstacle Detection &amp; Avoidance in Structured Environment</td>
<td>52</td>
</tr>
<tr>
<td>Proposed 3D Reconstruction Approach</td>
<td>53</td>
</tr>
<tr>
<td>Description of 3D Reconstruction Problem</td>
<td>54</td>
</tr>
<tr>
<td><strong>9 PROPOSED 3D RECONSTRUCTION IN LATERAL MOTION</strong></td>
<td>58</td>
</tr>
<tr>
<td>Equations of Camera Motion</td>
<td>58</td>
</tr>
<tr>
<td>Camera Vibration</td>
<td>99</td>
</tr>
<tr>
<td>Ego Motion Correction</td>
<td>101</td>
</tr>
<tr>
<td>Image Noise and its Handling</td>
<td>101</td>
</tr>
<tr>
<td>Implementation of all Proposed Solutions</td>
<td>103</td>
</tr>
<tr>
<td>14 CONCLUSION</td>
<td>115</td>
</tr>
<tr>
<td>APPENDIX A: DERIVATIONS FOR 3D RECONSTRUCTION EQUATIONS</td>
<td>117</td>
</tr>
<tr>
<td>APPENDIX B: OVERVIEW OF CHOSEN CAMERA (SONY FCB-EX980S) WITH IMAGE STABILIZATION</td>
<td>120</td>
</tr>
<tr>
<td>APPENDIX C: SELECTING APPROPRIATE VIBRATION INSULATORS</td>
<td>123</td>
</tr>
<tr>
<td>APPENDIX D: OVERVIEW OF MOTION ESTIMATION IMPLEMENTATION PLATFORM</td>
<td>128</td>
</tr>
<tr>
<td>APPENDIX E: IMPLEMENTED C++ CODE FOR PROPOSED MOTION ESTIMATION</td>
<td>131</td>
</tr>
<tr>
<td>APPENDIX F: PUBLISHED / SUBMITTED RESEARCH CONTRIBUTIONS</td>
<td>149</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>151</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1: Summary of Relevant Color Space Analysis</td>
<td>22</td>
</tr>
<tr>
<td>Table 2: Results of Multiple Thresholding Test Algorithm</td>
<td>27</td>
</tr>
<tr>
<td>Table 3: Values Used for Lateral Flight Simulations</td>
<td>70</td>
</tr>
<tr>
<td>Table 4: Values Used for Forward Flight Simulations</td>
<td>75</td>
</tr>
<tr>
<td>Table 5: Decision Matrix Based on Multi Criteria Decision Analysis</td>
<td>88</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1: GTAR Lama Aircraft Camera and Experimental Setup Used 12
Figure 2: Sample Input Images 13
Figure 3: Grayscale Histogram with Histogram Equalization Along With Histograms for Separated Red, Blue and Green Channels 14
Figure 4: Histograms and Laplacian Filtering for Various Color Spaces on One Test Image 15
Figure 5: Sample Cropped Images of LEDs 18
Figure 6: Color Space Analysis for Blue LED Image Group 19
Figure 7: Color Space Analysis for Green LED Image Group 20
Figure 8: Color Space Analysis for Yellow LED Image Group 20
Figure 9: Color Space Analysis for Red LED Image Group 21
Figure 10: Linear Confidence Level Calculation for a Chosen Parameter 30
Figure 11: Overall Confidence Level Calculation on 27 Test Images Containing Different LEDs 31
Figure 12: Overall Block Diagram of GTAR Lama Aircraft System 33
Figure 13: GTAR Lama Aircraft 35
Figure 14: Noisy Images 41
Figure 15: Final Implemented Algorithm 44
Figure 16: Output Images from HITL Tests 45
Figure 17: Output Images from Flight Tests 45
Figure 18: Overview of a Typical Computer Vision Problem 49
Figure 19: Georgia Tech GTMax UAV 54
Figure 20: Camera Mounted on a UAV with a Detected Object in the Scene 59
Figure 21: The Residual Vector on Image Plane 61
LIST OF SYMBOLS

$\Delta$Position  

$\Delta$Velocity, $\Delta$Orientation, $\Delta$AngRate  

$Q$Position  

$Q$Velocity, $Q$Orientation, $Q$AngRate  

$\phi, \theta, \psi, \phi', \theta', \psi'$  

$L_{bv}$  

$J$  

$C$  

$P$  

$Z$  

$x_k$  

$C_X, C_Z$  

$d$  

$f$  

$y_k, z_k$  

$X_{ck}, Y_{ck}, Z_{ck}$  

$dx, dy$  

$H_k$  

$R_k$  

$I_2$  

$a$  

$t$  

$A$  

position error

velocity, orientation & angular rate errors

process noise covariance (for position)

respective process noise covariances

orientation and orientation rates of the camera

rotation matrix from vehicle frame to body frame

error index for z-test correspondence

variance for error index

estimation error covariance matrix

projected measurement vector onto the image plane

projected database corner vector onto the image plane at step k

components of variance for error index

residual vector in image plane

focal length of the camera

2D screen coordinates of projected database corner vector $x_k$ at step k

3D coordinates of position of the camera at step k

components of residual vector in image plane

observation matrix for extended Kalman filter

covariance of the observation error

a 2x2 identity matrix

forward velocity

time in seconds

a real matrix of n rows and m columns
LIST OF ABBREVIATIONS

3D  Three Dimensional
AUUVSI  Association of Unmanned Vehicle Systems, International
BMA  Block Matching Algorithm
CCF  Cross Correlation Function
EKF  Extended Kalman Filter
FOE  Focus of Expansion
FOV  Field of View
GTAR  Georgia Tech Aerial Robotics
GTMax  Georgia Tech RMax Helicopter UAV
HITL  Hardware in the Loop
LED  Light Emitting Diode
MAE  Mean Absolute Error
MCDA  Multi Criteria Decision Analysis
MGDS  Modified Gradient Descent Search
MPEG  Moving Pictures Expert Group
MSE  Mean Squared Error
NED  North-East-Down
RGB  Red-Green-Blue color space
SEA  Successive Elimination Algorithm
UAV  Unmanned Air Vehicle
UAVRF  Unmanned Air Vehicle Research Facility (Georgia Tech)
SUMMARY

Efficient and effective vision systems are proposed in this work for object detection for ground & aerial robots venturing into unknown environments with minimum vision aids, i.e. a single camera. The first problem attempted is that of object search and identification in a situation similar to a disaster site. Based on image analysis, typical pixel-based characteristics of a visual marker have been established to search for, using a block based search algorithm, along with a noise and interference filter. The proposed algorithm has been successfully utilized for the International Aerial Robotics competition 2009.

The second problem deals with object detection for collision avoidance in 3D environments. It has been shown that a 3D model of the scene can be generated from 2D image information from a single camera flying through a very small arc of lateral flight around the object, without the need of capturing images from all sides. The forward flight simulations show that the depth extracted from forward motion is usable for large part of the image. After analyzing various constraints associated with this and other existing approaches, Motion Estimation has been proposed. Implementation of motion estimation on videos from onboard cameras resulted in various undesirable and noisy vectors. An in depth analysis of such vectors is presented and solutions are proposed and implemented, demonstrating desirable motion estimation for collision avoidance task.
CHAPTER 1

INTRODUCTION

Overview

Vision systems have great potential for a successful, collision-free motion of a ground or aerial robot through 3D space. Amongst many situations in which ground or aerial robots may be preferred over humans, at least initially to gather enough information, the most significant are unknown environments. In this dissertation, efficient and effective vision systems are proposed for object detection for ground and aerial robots venturing into such unknown environments, with minimum possible vision aids onboard, i.e. a single camera.

Part one deals with the problem of object detection in situations typical of disaster sites. Based on various image analysis techniques, including color histograms, filtering techniques and color space analyses, typical pixel-based characteristics of visual markers to be detected are established. A block based search algorithm is then used to search for those established characteristics in real-time image data stream from a color camera. After analysis of excessive noise encountered in the actual flight tests, a noise and interference filter was devised and applied. The final algorithm that was implemented on Georgia Tech Aerial Robotics Lama aircraft, used both multiple thresholding and linear confidence level calculations and was successfully utilized in the AUVSI’s International Aerial Robotics Competition 2009 to automatically detect a specified object.
Part two deals with object detection for ground and aerial robots for collision avoidance in 3D environments. Most common approaches to this problem include use of active sensors for 2D planar motion of ground robots, or using multiple active or passive sensors, or by developing a prior expectation map of the world and its comparison with the new image data to detect obstacles in 2D. In this work, the general ‘Structures from Motion’ or 3D Reconstruction problem has been improved upon. The equations developed and the simulation results presented, show that a 3D model of the scene can be generated from 2D image information from a single camera flying through a very small arc of lateral flight around the object, without the need of capturing images from all sides. The forward flight simulation results show that the depth extracted from forward motion is usable for large part of the image.

Besides other constraints, the computational effort involved in the above-mentioned 3D obstacle detection problem is tremendous, especially with the increase in number of feature points, as in a typical real world scene. Hence attempts to overcome such constraints have been investigated. Approaches like optical flows and flow field divergence have been considered and analyzed, but these too have inherent constraints. A new technique of using motion estimation for obstacle detection is proposed here. Motion estimation is used extensively for almost all video compression standards, including MPEG 4. Here, motion estimation has been proposed to solve the problem of object detection for collision avoidance. Implementation of this approach on videos recorded from a UAV revealed issues of image noise, interference and platform vibrations/jitter while in flight. An in-depth analysis of such issues is presented next and various solutions are proposed. These proposed solutions have been successfully
implemented and the results demonstrate the effectiveness of the proposed approach, vis-à-vis other approaches used to solve this collision avoidance problem.

**Summary of Contributions**

The contributions of this work are summarized below:

1. An approach has been developed for search and identification of a visual marker from a single camera. Using various image analysis techniques, such characteristics of a visual marker have been established, which are sufficient to successfully identify the desired visual marker in a real-time video stream.

2. A methodology has been proposed and implemented, in order to effectively handle noise and interference encountered in onboard videos from a flying UAV, without significantly increasing the computational overhead.

3. The proposed work has been successfully implemented for detecting a blue-LED from a real-time UAV video during the International Aerial Robotics Competition 2009.

4. For the problem of object detection for collision avoidance using a single camera, existing solution to a general Structures From Motion problem has been significantly improved by marked reduction in the path of flight required.

5. Object detection has also been successfully achieved while flying forward towards the objects in the scene. Earlier research considered information from such forward flight, to be unusable for large part of the image.

6. To better solve the single-camera based collision avoidance problem, Motion Estimation technique has been proposed, which attempts to overcome different constraints of 3D scene reconstruction, Optical Flows, Flow Field Divergence or
other approaches, and yet retains most of the merits of these approaches. This approach is though common to video compression, has not yet been applied to such a collision avoidance problem.

7. The problem of undesirable motion vectors in real videos from a UAV has been thoroughly investigated. Various solutions have been proposed, which have been successfully implemented. The results have demonstrated that motion estimation technique can be effectively utilized to detect new obstacles entering the scene, as well as to identify critical obstacles to be avoided, from the ones already present in the scene.

**Organization of Dissertation**

The rest of the dissertation is organized as follows:

Part I deals with the search and identification problem using a single camera and comprises of Chapters 2 to Chapter 7. The problem is introduced in Chapter 2 and relevant research in the area is presented. Preliminary experiments and investigation for object detection is presented in Chapter 3. This leads to a detailed analysis as presented in Chapter 4, in which two algorithms are proposed with respective test results. Implementation of the chosen algorithm on Georgia Tech’s participating vehicle into International Aerial Robotics Competition 2009 is presented in Chapter 5. Analysis of the noise encountered in real-time videos is presented in Chapter 6, which also includes proposed solutions and their implementation. The efficacy of such solutions is demonstrated via flight test results. A conclusion for part one is presented in Chapter 7.

Part II deals with the object detection problem for collision avoidance using a single camera. It comprises of Chapters 8 thru 13. An overview of related research is
presented in Chapter 8, followed by presentation of proposed 3D reconstruction approach while in lateral flight in Chapter 9. Object detection is also attempted while flying forward and this approach is presented in Chapter 10. Analysis of 3D reconstruction approach is presented in Chapter 11. Practical constraints associated to various techniques including 3D reconstruction, Optical Flows and Flow Field Divergence, are presented next in Chapter 12, after which Motion Estimation technique is proposed. The undesirable and noisy vectors encountered in Motion Estimation implementation are analyzed in detail in Chapter 13. Various solutions are proposed and implemented and successful results are presented.

An overall conclusion of the work presented in this dissertation is presented in Chapter 14.

Lastly, some pertinent information closely associated to the text in the dissertation is presented in Appendices A thru F. This includes derivations of various equations and results, implementation software, procedure for choosing requisite implementation hardware etc.
PART I

OBJECT DETECTION

USING IMAGES FROM A SINGLE CAMERA FOR

SEARCH AND IDENTIFICATION
CHAPTER 2
INTRODUCTION TO SEARCH & IDENTIFICATION PROBLEM
(PART I)

Various roles of ground and aerial robots include disaster management tasks during fires, earthquakes, floods, landslides, volcano eruptions, storms, etc., as well as for military or similar applications, such as reconnaissance, target identification, rendezvous, nuclear, chemical, biological and conventional warfare, etc.[1]. For all such roles, a ground or an aerial robot essentially requires some kind of environment or scene sensing, which directly leads us to the interest in vision-based systems, as vision sensors provide a wealth of information about the surrounding environment.

Vision based object detection and identification is also motivated by nature. Most animals employ some kind of vision system for object detection and identification. Humans for example, use two eyes, which are remarkable stereo vision sensors, giving us sufficient scene information for many purposes. Likewise, almost all animals use eyes as vision sensors to identify and locate objects. Vision systems for ants and bees are even known to gain the direction to important places from polarization patterns of the blue sky [2].

Although a ground or an aerial robot may employ various sensors like laser range finders, infra-red sensors, ultraviolet sensors, sonar, radar, thermal imaging sensors, etc., cameras are mostly used as the primary vision system sensors. In fact, cameras are one of the most general sensors for ground and aerial robots, since these deliver richer and more complete information than other sensors. Hence the aim of this work is to utilize the same camera, not just for taking images or recording video, but also to accomplish a
much more sophisticated task of searching, locating and identifying a specific visual marker in a video being recorded in real-time for unknown disaster-like environments.

It is aimed here to design a low cost, visual marker search and identification system for ground and aerial robots. A generic rule based model is developed to search and identify the specific visual marker in the presence of noise and interference. The chosen implementation platform is an aerial robot developed by Georgia Tech’s Aerial Robotics (GTAR) team for entry into the International Aerial Robotics Competition 2009 organized by Association for Unmanned Vehicle Systems International [3].

The competition required creating a small aerial robot capable of fully autonomous flight through a confined environment such as a power plant. The vehicle was first required to enter the building through a 1 square meter opening from a designated launch area 3 meters away. The vehicle was then to search for a target area while avoiding obstacles such as walls, columns, and furniture. Upon locating the control panel which was to be identified by various blinking lights and an audible warning tone, a picture of a specific target gauge with 18 font size text, a still or video was to be relayed by means of a radio frequency (RF) signal with sufficient power to be received at a distance of 300 ft with a loss of 6 dB (to account for attenuation by the structure). The target gauge of interest was the only one with a non-blinking blue light emitting diode (LED) directly below it, which led us to choose our visual marker that was to be searched and identified, as a steady-on blue LED.

**Related Research**

The overall problem of vision based object detection in noisy environments may be broken down into three sub-problems of establishing color or shape characteristics for recognition and identification, followed by a search algorithm to locate those identification criteria, and finally using some effective noise and interference handling techniques. One of the very relevant examples which closely matches the problem here,
includes [4], in which the authors propose a generic rule based color model for fire pixel
detection using YCbCr color space (luma, blue-difference and red-difference chroma
space). This color space was chosen to separate the luminance from the chrominance for
effectively addressing the issue of illumination variations. A set of rules defined on the
Y, Cb and Cr color components, together with the developed chrominance model on the
Cb-Cr color plane, were used to detect the fire pixels in color images.

With reference to existing approaches, the three sub-problems or detection, search
and noise handling are discussed next.

**Color / Shape Recognition and Detection**

Existing research demonstrates successful color or shape recognition by using
different color spaces and various mathematical techniques. Relevant examples include
[5] in which authors propose a vision-based street detection algorithm to be used by small
autonomous aircraft in low-altitude urban surveillance. This algorithm uses Bayesian
analysis to differentiate between street and background pixels. Similarly in [6], shadow
detection of color aerial images based on a successive thresholding scheme has been
presented by the authors. In [7] authors propose object detection by a color histogram-
based fuzzy classifier with support vector learning. They use hue and saturation color
space histograms to detect desired features. In [8], road recognition based on color image
edge detection has been considered. They have used the ‘Lab’ color space (L*, a*, b*
CIELAB color space of lightness and a & b color components, known to approximate
human vision), along with Hough transforms to extract edges of the road. In [9], traffic
sign shape recognition using geometric matching has been proposed. To reduce digital
noise and extract the shape of each individual traffic sign, the external boundaries of
traffic signs have been segmented based on color information and simplified and
decomposed through discrete curve evolution.
Keeping in view all these approaches, various color spaces were analyzed and the approach that finally developed in this dissertation had been comparable to [4] in that a rule based model based on various color spaces was developed for the visual marker in question. This model was then used to search for and identify the visual marker in the video stream.

**Block Based Matching and Searching**

In a block based approach two important aspects are that of what matching criteria is being employed and what search pattern is being used. The matching criteria could be that of an ‘exact’ match or a ‘best’ match. The most common example of an exact match is optical flow [10, 11], where an intensity constraint equation, along with a smoothness constraint, is used to find an exact matching pixel in the next image in time. This has been used to solve problems like obstacles avoidance, centering between obstacles, ego-motion calculation etc. for ground or aerial robots [12-16].

The typical case of a ‘best’ rather than an ‘exact’ match is that of motion estimation, in which some error such as mean absolute error, or mean squared error, is either minimized, or while using a cross-correlation function, the correlation is maximized [17-19]. This is very commonly used in video compression codecs such as motion pictures experts group’s MPEG-1 to MPEG-4 [20-22].

Various search approaches are employed in the literature to find the match as per the matching criteria chosen. Examples are [23-26], which give fast searching options. However, these do come with some information loss as the whole search space is not looked into, and an optimum search path is chosen so as to minimize the computational effort.

In this dissertation, based on simulations and flight test, a full or exhaustive search has been chosen, instead of employing an optimal search approach to avoid any possibility of missing out the visual marker / beacon. Typical example of such a full
search approach is [27] in which motion estimation has been done with a modified
gradient-descent search (MGDS) algorithm utilizing an adaptive computation distribution
mechanism. Similarly in [28] a hierarchical design methodology for full-search block
matching motion estimation has been proposed which decomposes a 2-D full search
problem into several 1-D functional blocks. Yet another example is that of [29], in which
a full search successive elimination algorithm (SEA) for variable block-based motion
estimation of MPEG/H.264 standard has been proposed.

The approach described in this dissertation uses fixed as well as variable grid
sizes on the image in a linear full search block based algorithm.

Colored Noise Analysis & Filtering

For noise handling, [30] was considered, which proposes color image noise
removal to eliminate noisy pixels by exploiting several vector-class characteristics of
multi-channel pixels. This algorithm treats multi-channel images as a vector class and
takes both magnitude and phase angles of the pixel vectors into consideration. Another
approach is that of [31], in which white and color noise cancellation using an adaptive
feedback cross-coupled line enhancer filter, is proposed. This method consists of two
adaptive FIR (finite impulse response) filters, where the output of each filter is fed back
to the input of the other filter to form an adaptive feedback cross-coupled filter.

Based on the image processing and analysis of the noise practically encountered,
the approach described here utilizes a very low cost thresholding noise filter which has
been demonstrated to work very effectively even for as high as 40% noisy pixels out of
the total pixels in the image data.
CHAPTER 3
PRELIMINARY EXPERIMENTS AND INVESTIGATION

To establish the characteristics of the visual marker or a beacon to be searched, a low cost camera was chosen for experimentation, which could also be later utilized on the actual implementation platform. An experimental setup was devised, as shown in Fig. 1. It comprised of a Panasonic CCTV color video camera, a wireless transmitter, a receiver with a VCE-Pro PCMCIA video capture card / frame grabber, a power supply, various LEDs as potential visual markers or beacons of interest, and a computer to record, store and process images in real time. For this testing a 2.4GHz wireless link was used here. Images were taken, varying from 1 ft to 6 ft distances from the test visual markers. Beyond 6 ft, the target test gauge would have been unreadable. Quite a few images were taken of red, blue, yellow and green LEDs along with other images with all, or no LEDs at all. Separate images of just a blue LED were taken in order to perform additional analysis on such images. Some sample input images are shown in Fig. 2.

Fig. 1. GTAR Lama Aircraft Camera and Experimental Setup Used. Panasonic CCTV Board Camera with a transmitter and a power supply, a receiver with VCE-Pro PCMCIA video capture card, various LEDs, and a computer for image recording and real-time processing.
Fig. 2. Sample Input Images. 27 such test images were used for preliminary analysis.

Grayscale Preliminary Analysis

Intensity transformation functions based on information extracted from image intensity histograms play a basic role in image processing in the areas of enhancement, compression, segmentation and description [32]. Besides looking at intensity histograms of our test images similar to [7], histogram equalization was also performed here, which is a technique to achieve enhancement by spreading the levels of the input image over a wider range of the intensity scale [32]. Hence the histograms were plotted both for the grayscale image and the equalized image of all test images. Results for a sample image are presented in Fig. 3. This grayscale analysis however, did not reveal any quantifiable information for us to detect the chosen beacon as the blue LED, or to separate an image with a blue LED from an image without a blue LED. Hence the histogram analysis was next performed, separately on the red, blue and green channels of the images (Fig. 3).
Fig. 3. Grayscale Histogram With Histogram Equalization Along With Histograms for Separated Red, Blue and Green channels. Similar analysis was done for all 27 test images.

Analysis of Color Spaces

Starting from RGB (red, green, blue) color space analysis, similar analysis was performed on quite a few other color spaces for the test images. This included the following spaces (refer Fig. 4 for a sample test image analysis):

- NTSC color space for luminance-Y, hue-I and saturation-Q channels
- YCbCr color space for luminance-Y, blue difference-Cb and red difference-Cr channels
- HSV color space for tint-H, shade-S and tone-V channels
- CMY color space for cyan, magenta and yellow channels
- HSI color space for hue, saturation and intensity channels
After a similar analysis was performed on all captured test images, it was observed that there were significantly more pixels with high blue channel values compared to the number of pixels with high red or green channel values in images containing the desired visual marker (blue LED), than the number of such pixels in images without a blue LED. Thus, for example, if a threshold level is chosen for a color component as $L_{th}$ and the number of pixels with higher levels of blue, red and green than this $L_{th}$ are counted for a test image, i.e.,

- For pixels with blue level  $B > L_{th}$, count all such pixels totaling to $N_B$,
- For pixels with red level  $R > L_{th}$, count all such pixels totaling to $N_R$,
• For pixels with green level \( G > L_{th} \), count all such pixels totaling to \( N_G \), then if

\[
\begin{align*}
\text{• } N_B > N_R \text{ such that } N_B / N_R &\approx 1.5 \quad \text{and} \\
\text{• } N_B > N_G \text{ such that } N_B / N_G &\approx 2.0
\end{align*}
\]

Then the test image contained a blue LED

(On a 320x240 image with 8-bit image pixels, where pixel values vary from 0-255 for each of the three channels, an \( L_{th} \) of around 200 was found to be a good threshold for distinguishing between images with or without the blue LED for the test distance of 1ft-6ft).

A test code was implemented to examine this observation. It was found that this code resulted in 80% success on the chosen test set of 27 images. Once the code was tested against many more images outside the chosen test set, containing bright blue color from sources other than LEDs, the code gave false positives. This was for obvious reason of having a large number of high blue channel valued pixels in such images, which would give \( B > L_{th} \) as well as \( N_B / N_R \approx 1.5 \) and \( N_B / N_G \approx 2.0 \). Hence, it was concluded that the chosen beacon could not be detected with practical certainty using this method.

**Analysis Based on Camera Filters**

The use of a blue filter over the lens of the camera was also explored. Images were captured using one layer of the filter as well as two layers. Upon analysis of such images and testing against the above method, any additional facts or observations could not be concluded.

Image enhancement using Laplacian filtering [33] was also explored, which is based on the following equation,
\[ g(x,y) = f(x,y) + c[\nabla^2 f(x,y)] \]

where \( f(x,y) \) is the input image, \( g(x,y) \) is the enhanced image and \( c \) is 1 if the center coefficient of the mask is positive, or -1 if it is negative. A 3x3 Laplacian sharpening mask was used as follows (since center coefficient of this mask is negative, hence \( c=-1 \) is chosen for the above equation):

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & -8 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

The result on one test image is shown in Fig. 4 above (last row). However, this filtering also did not give us any conclusive information.

Hence, only the specific LED parts from the images were cropped out next for consideration, and a detailed analysis followed.
CHAPTER 4

DETAILED ANALYSIS AND PROPOSED BLOCK BASED SEARCH ALGORITHMS

Detailed Analysis

For the detailed analysis of cropped out images of specific LED signatures in images, the goal was to establish a generic rule based model for the desired visual marker as seen from our camera in an image (similar to [4], in which similar methodology was used to detect fire in the images). Hence not only a blue LED was analyzed here, but also other images including red, green and yellow LEDs were evaluated, so as to clearly differentiate a blue LED from other LEDs. 59 such images were chosen with just the LEDs showing, including 29 images of blue LEDs, 8 of green, 7 of yellow and 15 of red LEDs for this analysis. These images were taken under different lighting conditions (refer Fig. 5 for some sample images).

Fig. 5. Sample Cropped Images of LEDs. 59 such images were used in total, including 29 images from blue LED group, 8 from green LED group, 7 from yellow LED group and 15 from red LED group.
The following nine parameters for each of the cropped images were calculated:

8. Average value of red level of a pixel over the cropped image (showing the zoomed-in LED).
9. Average value of blue level of a pixel over the same cropped image.
10. Average value of green level of a pixel over that image.
11. Average value of intensity of a pixel of that image.
12. Ratio between average red level and average intensity level of that image.
13. Ratio between average green level and average intensity level of that image.
14. Ratio between average blue level and average intensity level of that image.
15. Ratio between average blue level and average red level of that image.
16. Ratio between average blue level and average green level of that image.

The results from this analysis for each of the four groups i.e. the blue LED image group, the green LED image group, the yellow LED image group and the red LED image group, are shown in Figs. 6, 7, 8, & 9 respectively.

![Graph](image)

**Fig. 6. Color Space Analysis for Blue LED Image Group.**  
*a)* shows average red, green, blue and intensity levels in pixels belonging to a blue LED image.  
*b)* shows ratios between these channels. Pixel RGB and intensity values are calculated by averaging over all the pixels in each of the cropped images showing the zoomed-in LED.
Fig. 7. Color Space Analysis for Green LED Image Group.  

a) shows average red, green, blue and intensity levels in pixels belonging to a green LED image.  
b) shows ratios between these channels. Pixel RGB and intensity values are calculated by averaging over all the pixels in each of the cropped images showing the zoomed-in LED.

Fig. 8. Color Space Analysis for Yellow LED Image Group.  

a) shows average red, green, blue and intensity levels in pixels belonging to a yellow LED image.  
b) shows ratios between these channels.Pixel RGB and intensity values are calculated by averaging over all the pixels in each of the cropped images showing the zoomed-in LED.
Fig. 9. Color Space Analysis for Red LED Image Group.  

*a*) shows average red, green, blue and intensity levels in pixels belonging to a red LED image.  
*b*) shows ratios between these channels. Pixel RGB and intensity values are calculated by averaging over all the pixels in each of the cropped images showing the zoomed-in LED.

For the respective LED image groups, a summary of the above data analysis in terms of observed minimum, maximum and average values along with standard deviations, is presented in Table 1. This gives an idea of typical characteristics of these LEDs as seen in our experimental setup.
TABLE 1. Summary of Relevant Color Space Analysis. There are 29 images in the blue LED image group, 8 in green LED image group, 7 in yellow LED image group and 15 in red LED image group.

<table>
<thead>
<tr>
<th>IMAGE GROUPS</th>
<th>Average Channel Values of an LED Image</th>
<th>Ratios between Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg R (Red)</td>
<td>Avg G (Green)</td>
</tr>
<tr>
<td>Blue LED</td>
<td>Min 186</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>Max 254</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td>Avg 216</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Std Dev 18.36</td>
<td>20.83</td>
</tr>
<tr>
<td>Green LED</td>
<td>Min 192</td>
<td>196</td>
</tr>
<tr>
<td></td>
<td>Max 253</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>Avg 216</td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>Std Dev 18.92</td>
<td>16.41</td>
</tr>
<tr>
<td>Yellow LED</td>
<td>Min 188</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td>Max 244</td>
<td>236</td>
</tr>
<tr>
<td></td>
<td>Avg 206</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>Std Dev 17.37</td>
<td>15.75</td>
</tr>
<tr>
<td>Red LED</td>
<td>Min 180</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Max 255</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>Avg 243</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>Std Dev 20.22</td>
<td>36</td>
</tr>
</tbody>
</table>
For detecting the blue LED, those characteristics of this chosen marker were identified, which were not common to other LEDs. Hence from the above analysis (presented in Figs. 6 to 9 and Table 1), the following are deduced:

1. From the blue LED image group, a minimum average blue level ($Min_B$) could be defined for all images in that group. We call it the ‘minimum average’, as first, an average level is found over one image for all pixels in that image, then a minimum is found from the set of all such values for each of the images in that group; hence the name ‘minimum average’ level for that group. This $Min_B$ value was observed to be significantly higher for blue LED image group than for all the other LED image groups. Referring to Table 1, it is 221 for the blue LED image group, as compared to 142, 152 and 108 in the green, yellow and red LED groups respectively. (Also refer to Figs. 6 to 9.)

2. A minimum intensity level ($Min_I$) could also be defined, which is observed to be significantly higher for the blue LED image group as compared to the green, yellow or red LED image groups. In Table 1, it is 200 for blue LED group as compared to 179, 180 and 124 for green, yellow and red LED image groups respectively. (This would however, be true for a white LED as well, and hence cannot be used as a stand-alone criteria to identify the desired visual marker/beacon.)

3. The average value of blue content (averaged over all pixels in an image, followed by averaging over such values for all images in that group) in the blue LED image group is always significantly higher than other colors. This behavior (which is otherwise intuitive), is not observed in all the other LED image groups and hence
is identified as a peculiar characteristic of the chosen beacon. This value is observed to be 247 for the Blue LED image group, as opposed to 176, 166 and 193 respectively for the green, yellow and red LED image groups (refer Table 1).

4. Referring to Table 1 again, with $B/R$ being the ratio between average blue and average red content per pixel, the minima, maxima, average and standard deviations are presented for all such $B/R$ values over the entire groups. Thus within the blue LED image group, the minimum of $B/R$ for the group is $Min_{B/R}$ which is observed to be at least 1.000 (Table 1). This value is much higher than the similar minima of the ratio $Min_{B/R}$ for the green, yellow and red LED image groups (0.741, 0.706 and 0.602 respectively in Table 1). This observation supplements observation 3 above.

5. Similarly observing the $B/G$ values, which is the ratio between average blue levels and average green levels per pixel, the minima for the blue LED image group $Min_{B/G}$ is observed to be 1.024. This value is also significantly higher than $Min_{B/G}$ values for the green, yellow and red LED image groups (0.695, 0.723 and 0.931 respectively in Table 1). Like observation 4 above, this deduction also supplements observation 3 above.

6. In the same way, comparing average blue levels with average intensity levels per pixel within the blue LED image group, the minima for $B/I$ values for this group $Min_{B/I}$ is observed to be 1.008. This value is significantly higher than similar minimas $Min_{B/I}$ values for the green, yellow and red LED image groups (0.792, 0.789 and 0.816 respectively in Table 1).
Multiple Thresholding Algorithm and Testing

Based on the findings from the image analysis presented above, the first approach that was considered, was that of using multiple thresholds. A block based search algorithm was utilized to find a block of pixels in an image which passes all the chosen thresholds. Such a block of pixels in that image is most probably a block belonging to the chosen visual marker. Keeping in view data summary from Table 1 and relevant discussion above, following fixed thresholds could be chosen

\[ \text{Min}_B \]
\[ \text{Min}_I \]

where the threshold values are based on the above data analysis. For implementation in our case, these values are chosen as 220 and 200 respectively, for the blue LED based on above findings. The other three thresholds chosen were \( \text{Min}_{B/I} \), \( \text{Min}_{B/R} \) and \( \text{Min}_{B/G} \). However, the constant values to be used for these three thresholds were found by repeated iterations as discussed below.

The implementation of this algorithm required that a block of pixels passes all these five thresholds in order to be identified as the most probable one belonging to the desired beacon to be found. An initial Matlab implementation comprised of a linear full search block matching algorithm with block size kept at 8x8 pixels in a fixed grid. We were scanning in raster order, to find the 8x8 block of pixels in the image that would pass all the five chosen thresholds. For this implementation various combinations of \( \text{Min}_{B/I} \), \( \text{Min}_{B/R} \) and \( \text{Min}_{B/G} \) were attempted with the following results on the initial test set of 27 sample images (the set contained a total of 27 images with 17 images containing the desired visual marker - blue LED and ten other images not containing it).
1. Choosing minimum observed values for $\text{Min}_{B/I}$, $\text{Min}_{B/R}$ and $\text{Min}_{B/G}$ thresholds calculated from the blue LED group (i.e. 1.008, 1.000 and 1.024 from Table 1, respectively), the multiple thresholding block search algorithm identified all 17 blue LED images out of 27, but gave 9 false positives, which was unacceptable.

2. Choosing minimum plus one standard deviation values for $\text{Min}_{B/I}$, $\text{Min}_{B/R}$ and $\text{Min}_{B/G}$ thresholds calculated from the blue LED group (i.e. $1.008 + 0.054 = 1.062$, $1.000 + 0.085 = 1.085$ and $1.024 + 0.102 = 1.126$ respectively, from Table 1 above), 16 blue LED images out of 17 were correctly identified, but gave 5 false positives out of 9. This was significantly better than the ones obtained by just choosing minimum values, as in the first case above.

3. Choosing minimum plus 2 times standard deviation values as the thresholds calculated from the blue LED group (i.e. 1.115, 1.169 and 1.228), correctly identified 14 blue LED images out of 17 and did not give any false positives out of the rest 9 images. This was a further improvement from the previous step 2.

4. Choosing mean values calculated from the blue LED group (i.e. 1.121, 1.151 and 1.244 from Table 1) also correctly identified 14 blue LED images out of 17 and did not give any false positives. The success rate in this case was the same as case 3 above.

5. Choosing minimum plus 3 times standard deviation values as the thresholds (i.e. 1.169, 1.254 and 1.33), identified only 9 blue LED images out of 17 but did not give any false positives. These thresholds were more stringent than the cases 3 or 4 above.
6. Choosing maximum values for $\text{Min}_{B/I}$, $\text{Min}_{B/R}$ and $\text{Min}_{B/G}$ thresholds (i.e. 1.2, 1.3 and 1.39), correctly identified only 5 of the 17 total blue LED images and did not give any false positives. These were the most stringent thresholds and would only pick very bright looking blue LEDs in the images.

7. A few more iterations of the code were run to find some empirical values for identifying all 17 blue LED images avoiding any false positives. These values came out to be 1.107, 1.08 and 1.228 respectively, for $\text{Min}_{B/I}$, $\text{Min}_{B/R}$ and $\text{Min}_{B/G}$ thresholds.

The above results are summarized in Table 2 below.

Table 2. Results of Multiple Thresholding Test Algorithm. The test set contained 27 images, with 17 images showing the visual marker and 10 images without it.

<table>
<thead>
<tr>
<th>Chosen criteria for thresholds $\text{Min}<em>{B/I}$, $\text{Min}</em>{B/R}$ and $\text{Min}_{B/G}$ from blue LED image group</th>
<th>Results for correct visual marker identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set thresholds to : $\text{Min}$ values</td>
<td>17 out of 17 correctly identified, 9 false positives</td>
</tr>
<tr>
<td>Set thresholds to: $\text{Min} + 1 , \sigma$</td>
<td>16 out 17 identified, 5 false positives</td>
</tr>
<tr>
<td>Set thresholds to: $\text{Min} + 2 , \sigma$</td>
<td>14 out of 17 identified, 0 false positives</td>
</tr>
<tr>
<td>Set thresholds to: $\text{Mean}$ values</td>
<td>14 out of 17 identified, 0 false positives</td>
</tr>
<tr>
<td>Set thresholds to: $\text{Min} + 3 , \sigma$</td>
<td>9 out of 17 identified, 0 false positives</td>
</tr>
<tr>
<td>Set thresholds to: $\text{Max}$ values</td>
<td>5 out of 17 identified, 0 false positives</td>
</tr>
<tr>
<td>Set thresholds to: Empirical values from repeated iterations</td>
<td>17 out of 17 correctly identified, 0 false positives</td>
</tr>
</tbody>
</table>

where $\sigma$ denotes standard deviation of the data.
It may however be said that these results were based on the 27 test images that were chosen initially, and for a block size of 8x8 pixels in a fixed search grid on the image. These thresholds, therefore, needed to be further tested not only with other block sizes, but also with variable search grids, and more importantly, in the final implementation of this algorithm with real time video from the onboard camera of our aircraft. This testing was required during both hardware in the loop and actual flight tests, in order to confirm the efficacy of the proposed algorithm.

**Linear Confidence Level Algorithm and Testing**

Apart from multiple thresholding proposed above, another algorithm is also proposed, which defines a confidence level for the visual marker or a beacon detection in an image. Referring to discussion in the previous chapter, this is not an ‘exact’ match case. It rather could be termed as a search for the ‘best’ match. Hence whatever percentage of matching is found on a block of pixels, it is recorded as a confidence level on the chosen beacon detection. At the end of searching within the current image, the best confidence level and the corresponding block found on this image is recorded. At the end of a whole video sequence, out of all such recorded best confidence levels found for each of the images, the maximum confidence level of the whole video stream is recorded as the final output from the algorithm. The corresponding image most likely contained the blue LED. It may be realized that this approach is much more robust than the approach of searching for the ‘exact’ match as per the search criteria.

Thus, for each of the five parameters considered i.e \( Min_B, \ Min_I, \ Min_{B/I}, \ Min_{B/R} \) and \( Min_{B/G} \), a confidence level is defined as follows,

\[
\text{Confd}_X_k = (\text{Confd}_X_{Max} - \text{Confd}_X_{Min}) \cdot (X_k - \text{Min}_X) / (\text{Max}_X - \text{Min}_X) + \text{Confd}_X_{Min}
\]
such that,

\[ Confd_{X_{\text{Max}}} \leq Confd_{X_k} \leq Confd_{X_{\text{Min}}} \quad \text{and} \quad Max_X \leq X_k \leq Min_X. \]

and where,

- \( X \) is any of the five chosen parameters i.e. \( Min_B, Min_I, Min_{B/I}, Min_{B/R} \) and \( Min_{B/G} \),
- subscript \( k \) is index of the block of pixels under consideration in the current image,
- \( Confd_{X_k} \) is the confidence level of having a blue LED for the \( k^{th} \) block under consideration in the current image,
- \( Confd_{X_{\text{Max}}} \) is the maximum confidence level corresponding to the maximum expected value \( Max_X \) for one of the five parameters \( Min_B, Min_I, Min_{B/I}, Min_{B/R} \) and \( Min_{B/G} \) (refer Fig. 10),
- \( Confd_{X_{\text{Min}}} \) is likewise the minimum confidence level corresponding to the minimum expected value \( Min_X \) of one of the five parameters \( Min_B, Min_I, Min_{B/I}, Min_{B/R} \) and \( Min_{B/G} \) (refer Fig. 10).

It may be noted that this is simply a straight line approximation for the confidence level calculation for \( k^{th} \) block of pixels under consideration for each of the five parameters \( X \) (i.e. any of \( Min_B, Min_I, Min_{B/I}, Min_{B/R} \) and \( Min_{B/G} \) ), as depicted in Fig. 10.

The values of \( Min_X, Max_X \), minimum confidence level \( Confd_{X_{\text{Max}}} \) and maximum confidence level \( Confd_{X_{\text{Min}}} \) are chosen based on the data analysis from Table 1. Thus if a block of pixels crosses the minimum threshold \( Min_X \) on this parameter, it is considered to be \( Confd_{X_k} \) probable as being a block belonging to the desired visual marker/beacon.
Similarly if it reaches the max threshold $Max_X$, we are $Confd_{X_{Max}}$ confident for this specific parameter that the block under consideration belongs to the desired visual marker.

After defining a confidence level for each of the chosen five parameters individually, an overall confidence level is defined for overall certainty of the block under consideration containing the desired visual marker. This is done by taking a product of confidence levels for each of the chosen five parameters. Hence,

$$Confidence_{Overall} = \prod_{j=1}^{5} Confd_{X_j} \quad (j=1,...,5 \text{ for five chosen parameters})$$

**Fig. 10. Linear Confidence Level Calculation for a Chosen Parameter.** $Confd_{X_k}$ is the confidence level of having a blue LED for the $k^{th}$ block under consideration in the current image. $Confd_{X_{Max}}$ is the maximum confidence level corresponding to the maximum expected value $Max_X$ for one of the five parameters $Min_B$, $Min_I$, $Min_{B/I}$, $Min_{B/R}$, and $Min_{B/G}$. $Confd_{X_{Min}}$ is likewise the minimum confidence level corresponding to the minimum expected value $Min_X$ of one of these five parameters.

The initial implementation of this algorithm thus comprised of a linear full search block matching technique, with block size kept at 8x8 pixels in a fixed grid. We were
scanning in raster order, to find the 8x8 block of pixels in the image that would give us the maximum overall confidence level \((\text{Confidence}_{\text{Overall}})\) of having the blue LED in that image. Thus on our test set, all the images containing the blue LED must give higher overall confidence levels than all the other images. The results of this implementation are as shown in Fig. 11 on the initial sample of 27 test images.

![Confidence Level for Blue LED detection](image)

**Fig. 11. Overall Confidence Level Calculation on 27 Test Images Containing Different LEDs.** Plot shows Linear Confidence level calculation for having a blue LED in all images. Almost all images containing blue LED give higher confidence levels than other LED images.

It may be observed that almost all the images containing the desired visual marker (blue LED) give higher confidence levels than other LED images. Further, no images have attained more than 0.9 confidence level. This is because \(\text{Min}_X, \text{Max}_X, \text{Confd}_X\text{Max}\) and \(\text{Confd}_X\text{Min}\) had been chosen based on the test image analysis and the proposed algorithm was being tested against the same test set. Hence we do not enter the dotted line portion of the straight line approximation of Fig. 11. For unknown images, the application of the same technique would give confidence level higher or lower than the
solid line portion of Fig.11. In such a case the relative overall confidence level should be significant rather than the specific numerical value.

Like the multiple thresholding case, it may be mentioned here again that these results were based on the 27 test images chosen initially and for a block size of 8x8 pixels in a fixed search grid on the image. This proposed algorithm therefore, needed to be further tested not only with other block sizes but also with variable search grids and more importantly, in the final implementation of this algorithm with real time video from the onboard camera of our aircraft.

In fact, this linear confidence level algorithm being much more robust than the earlier multiple thresholding algorithm, was the first algorithm implemented for HITL and flight tests.
An overall block diagram of the proposed system is presented in Fig 12 below. Onboard vision system comprises of a camera and a video transmitter and through a 900Mhz link, the video is received at the Ground Control Station for processing and searching an identifying the desired visual marker.

**Fig. 12. Overall Block Diagram of GTAR Lama Aircraft System.** The vision system uses 900Mhz wireless link, whereas the Guidance, Navigation and Control systems use 2.4 Ghz link.
Implementation Platform & Onboard Hardware

The aerial platform used for this research is a coaxial miniature rotorcraft, which inherently possesses a stick-free stability (Fig. 13). It has a pair of counter-rotating blades, making the vehicle more compact since no tail rotor is required for yaw control. The bottom set of blades has cyclic control for maneuvering, while the upper set of blades has a Bell stabilizer (also known as a flybar) to counteract vehicle pitch and roll, providing attitude stability. The attitude drift is bounded when the vehicle is properly balanced, allowing the design of controllers that do not require information about the system attitude or angular rate. The base helicopter selected for development was the E020 Big Lama [34], made by E-Sky. It has a rotor diameter of 46cm and weighs approximately 410g in its stock configuration. Initial flight tests indicated that the stock vehicle has a useful payload capacity of approximately 50g which was further increased by removing the canopy and upgrading the motors. The final flight configuration with avionics, a larger battery, brushless motors, and a protective shroud weighed 605g with a rotor diameter of 0.61 meter.
Fig. 13. GTAR Lama Aircraft. This aircraft was indigenously developed from E020 Big Lama [34] at UAVRF of Georgia Tech. The research work presented here was successfully implemented on it and led to its participation in AUVSI’s International Aerial Robotics Competition 2009.

Off-the-shelf range sensors were used to provide local position information. The MaxBotix LV-EZ1 sonar was chosen for measuring altitude, and the SHARP GP2Y0A02YK0F infrared sensor was selected for measuring range to the walls and other obstacles. Range measurements from the IR sensors were also used to estimate vehicle heading. An ATMega128 onboard microprocessor was used to read the sensors and process the data for navigation. The data from these sensors was processed and filtered appropriately, according to their error characteristics for simple obstacle avoidance and wall-following behavior. The IR range sensors having analog voltage outputs were read by onboard analog to digital input channels. The altitude sonar was read via serial port.
Data was transmitted to the ground via a wireless link using an additional serial port on the ATMega128. The sonar was mounted pointing downward. Two IR sensors were placed 45cm apart looking forward. These were for heading control, for improved obstacle avoidance during forward motion, for detection of openings such as windows or doors, and for wall-following behavior. In addition, one IR sensor each was pointed left and right to detect obstacles and oncoming walls during wall following flight.

For the vision system the same Panasonic CCTV Board Camera was used here, with a VCE-Pro PCMCIA video capture card / frame grabber, as was used during testing phase. Initially a 2.4GHz link was used for all equipment, but because of excessive interference from all other onboard equipment, the video link was switched to 900 MHz.

**Onboard Software Overview**

**Navigation Algorithm [35]**

The navigation algorithm comprised of a smart filtering routine to prevent large step or impulse inputs from adversely affecting the controller. A Kalman filter was used to estimate the range, and the covariance of the residuals was used to detect and ignore outliers beyond three standard deviations. Occasionally an actual discontinuity in the range would occur, such as flight over an obstacle on the ground. The smart filtering routine would recognize such events and would adjust the range estimate to match the new measurements without changing the velocity estimate.

**Guidance Algorithm [35]**

For guidance, the vehicle followed the walls by flying laterally while detecting and maneuvering around corners, using side-looking range sensors. A separate altitude-hold controller was used to maintain a fixed altitude throughout the flight. Initial entry into the building was handled by a “Window Entry” logic. In this mode, an object detected on the forward-looking left or right IR sensors caused the lateral controller to
adjust the flight path to remain centered on the window. Once the vehicle entered the competition arena, walls were detected by the left IR sensor and the vehicle entered “Left Turn” mode. In this mode, an open-loop left turn was commanded until the forward-looking IR sensors detected the wall and “Wall Follow” mode would begin. In this mode, the longitudinal controller maintained a commanded distance from the wall while the heading controller maintained the desired heading with respect to the wall. The vehicle then used an open loop lateral command to fly along the wall to the right, using the right facing sensor to detect walls and obstacles in the flight path. Once an obstacle or wall was detected by the right looking IR sensor, the different corner-turning modes were initiated. If a wall or obstacle was detected in the direction of flight, the vehicle entered “Inside Turn” mode, whereby it changed its heading to either turn the corner (for concave corners) or fly around the obstacle. This was achieved by giving an open-loop yaw command until no obstacle is seen by the right IR sensor. If one of the forward-looking IR sensors detected a step change to max range while the other sensor still read near the estimated wall distance, a convex or outside corner had been detected. The vehicle then entered “Outside Turn” mode and it yawed to the left in order to continue around the corner. Once an inside or outside corner had been turned, and valid range measurements were seen on the two front IR sensors, the vehicle returned to “Wall Follow” mode and continued flight until the entire indoor area was traversed.

Control Algorithm [35]

For the control algorithm, since the coaxial rotorcraft platform inherently possesses attitude stabilization, the controller only provides servo deflections such that the vehicle is able to track a commanded direction. Although the controller has access only to the local range information, by exploiting the fact that indoor structures have walls, the controller can be designed to follow the walls. This circumvents the requirement for a global position fix. Consequently, the control architecture used was a
proportional integral derivative (PID) design with gain scheduling applied, such that the controller used different gain values depending on where the vehicle is, with respect to the wall. A Kalman filter based local velocity estimator was used in lieu of measured data. The position command was then directly linked to the actuator deflection using a PID control logic. The control action was thus achieved by using four independent control loops, i.e. altitude hold, heading hold, longitudinal position control and lateral position control.

**HITL Tests and Flight Tests**

For hardware in the loop tests, only the aircraft camera was on, and the aircraft was being manually carried around to simulate the flight. At first, only the linear confidence level algorithm (Chapter 4) was implemented in C-code and integrated with the aircraft software. Depending on these tests, necessary tuning of $Min_X$ values was done for all five parameters: $Min_B$, $Min_I$, $Min_{B/I}$, $Min_{B/R}$ and $Min_{B/G}$ so as to get a satisfactory confidence level for detection of the blue LED and avoiding false positives from bright, or light blue colored objects in the scene. Unlike the Matlab implementations until now, various block sizes with variable search grids on the image were now experimented with, in this real time implementation.

The search grid was controlled using a variable ‘grid step’, with its value varying between 1 and the block size. A grid step of 1 would mean that a block would be picked for analysis from every pixel in the image. Thus the probability of finding the blue LED block would be high with a high overall confidence level, since all possible blocks would have been considered for analysis. A smaller grid step like 1 would, however, increase the computational load, as now the number of candidate blocks in an image is equal to the number of pixels in that image. On the other hand, a grid step value, as large as equal
to the block size itself, would mean a fixed grid on the image. This would make the algorithm very fast as very few calculations would be required on such a fixed grid. However, it would give much lesser overall confidence levels, especially for a case where a blue LED overlapped multiple blocks. The tests revealed that if a grid step of less than half of the block size was chosen, the computational load would increase to an extent that a real-time application would not be practical. Hence, the grid step was subsequently kept at half of the block size.

This ‘hardware in the loop’ testing was followed by flight tests with all the onboard aircraft equipment turned on, and the aircraft autonomously flying around. However, excessive colored noise, as high as 40% noisy pixels of the overall video stream, was observed in such images. Many of the images individually had even 90% of the image distorted or interfered by noise. This was further investigated as described in the following chapter.
CHAPTER 6
NOISE HANDLING AND FINAL ALGORITHM

Noise Analysis and Handling

The excessive noise that was encountered during the flight tests could have been due to the following [32]:

1. Interference with other avionics
2. Noise due to wireless link
3. Electronic noise in digital camera and circuitry
4. Shot noise of an ideal photon detector, etc.

This noise was giving false positives and was generating higher confidence levels than an actual blue LED. In order to handle it, various approaches as discussed in Chapter 2 previously, were considered. However, use of any such approach would have increased the computational load even further. We were already restricted to half the block size as the grid step due to computational load as discussed in the previous chapter. Hence such a technique to handle this excessive noise was required, that would not only be very low cost but also be very effective in a real-time scenario. Initially this seemed quite challenging, but a deeper analysis led to interesting observations on the noise being encountered. Some typical noisy images are shown in Fig. 14, with representative pixels marked at left bottom.
Fig. 14. Noisy Images. Up to 40% of our total digitized images in a video stream were noisy initially. On some of the individual images, noise was as high as 90-100%. Noisy pixels were found to be either very low on all three channels a), or were very high on one or two channels b), c) & d). Almost no noise was encountered that was high on all three channels. Cases b), c) and d) included blue channels and gave very high confidence levels (false positives) for our initial algorithm.

Further image analysis of various noisy images, which individually had up to 90% of the image distorted by noise, revealed the following,
1. The noise generated was either very low on all three channels (very low intensity),
2. Or, was very high on just one channel (any of red, blue or green channels),
3. Or was very high on any two of the three channels,
4. Almost no such noise was encountered that was high on all three channels (very high intensity).

Thus, whenever this noise was very high on either blue channel alone, or any two of the three channels (one being blue), it gave pretty high confidence levels, much more than the blue LED, leading to false positives. This desired visual marker therefore, could not be successfully detected in a given video stream in the presence of such noise.

This analysis of noise however suggested that if a minimum threshold is put on all three channels simultaneously, all noisy blocks of pixels could be avoided. Hence, if a block of pixels does not pass all these three thresholds simultaneously, it should not be considered as a candidate for detection of the desired visual marker. It may be realized that this thresholding for noise did not require any further calculations from the original proposed algorithm, as the three channels were to be calculated anyway for every block, for rest of the algorithm. Instead, implementing these thresholds for noise, in fact reduced the computational load as further calculations and processing for those noisy blocks of pixels was not required which did not pass all three thresholds.

Hence implementing the same, subsequent flight tests revealed successful detection of a blue LED without having false positives from noisy images or noisy blocks of pixels.
Final Implemented Algorithm

Based on hardware in the loop tests, flight tests and noise analysis, it was decided to use a mix of both multiple thresholding and linear confidence level algorithms. Hence, out of the five thresholds proposed in the multiple thresholding algorithm above (Chapter 4), the following minimum thresholds had already been chosen as base lines

1) $Min_B$
2) $Min_I$

Based on the noise analysis discussed above, two more thresholds were added as follows

3) $Min_R$
4) $Min_G$

After a block of pixels would pass all these four thresholds, linear confidence level calculations would be done for all the five parameters, exactly as described in Chapter 4. The final implemented algorithm is as depicted in Fig. 15.

After the tests were performed in the actual arena for International Aerial Robotics Competition, following values for the thresholds were finalized:

- $Min_B = 180$,
- $Min_R = 145$,
- $Min_G = 145$,
- $Min_I = 165$.

These values were optimized for 320x240 image resolution on an 8 bit image with pixel values varying 0-255, while the camera was at a distance of 1-6ft from the visual marker. For different image resolutions and distances of camera from the visual markers, suitable numbers need to be found by real-time testing.

Sample output images for this final algorithm from HITL tests are presented in Fig. 16 and from actual flight tests are presented in Fig. 17.
**Fig. 15 Final Implemented Algorithm.** After the tests were performed in the actual arena for International Aerial Robotics Competition, following values for the thresholds were finalized:

Min blue = 180, Min red = 145, Min green = 145, Min intensity = 165.

These values were optimized for 320x240 image resolution on an 8 bit image with pixel values varying 0-255, while the camera was at a distance of 1-6ft from the visual marker. For different image resolutions and distances of camera from the visual markers, suitable numbers need to be found by real-time testing.
Fig. 16. Output Images from HITL Tests. The images depict successful detection of blue LED with highest confidence level at the location of the blue LED within an image a), and picking the image with highest confidence level of all images in the whole video stream b). The numbers displayed on bottom left corners are the highest overall confidence level achieved. The relative confidence level values are significant here rather than the specific numerical values.

Fig. 17. Output Images from Flight Tests. The images depict successful detection of blue LED with highest confidence level at the location of the blue LED within an image a), and picking the image with highest confidence level of all images in the whole video stream b). Although the camera frame grabber could have provided images at 30 frames per second, these could be processed only at about 15 frames per second from the proposed algorithm. This was however, sufficient to detect the blue LED in real time.
CHAPTER 7
CONCLUDING SEARCH & IDENTIFICATION PROBLEM (PART I)

A vision-based algorithm has been presented here that detects a visual marker in real time and in the presence of colored noise and interference from the onboard avionics. Various image analysis techniques including color histograms, filtering techniques and color space analyses were utilized to establish typical pixel-based characteristics of the desired visual marker in an image. A block based search algorithm was used to search for those established characteristics in a real-time image data stream from a colored camera. A confidence level was defined based on all the significant criteria, so that higher the confidence level, higher the probability of having found the desired visual marker in the image stream. A very low cost noise and interference filter was implemented in order to handle excessive noise that was encountered during flight tests. This also efficiently eliminated the need for processing the unwanted or noisy pixel blocks, thus reducing computational costs significantly. The final algorithm used both multiple thresholding and linear confidence level calculation in a full search block based approach. This algorithm now is capable of handling as high a noise and interference as 30-40% in the image data. It was implemented on the GTAR lama aircraft and was successfully used in International Aerial Robotics competition 2009 (by AUVSI) to detect a blue light emitting diode in real time.

The approach presented here is utilizable for any future vision based object detection task in a real time video stream and in the presence of noise and interference. Example applications are vision based target identification, rendezvous, beacon finding, reconnaissance, disaster area surveillance etc.
PART II

OBJECT DETECTION

USING IMAGES FROM A SINGLE CAMERA FOR

COLLISION AVOIDANCE
A vision system as defined by Marr in [36] is a “Process that creates, given a set of images, a complete and accurate representation of the scene and its properties”. This definition is considered ‘general vision’, as the extracted representation of the scene has to be as general as possible. There are two approaches to scene representation. First is an accurate and complete representation of an observed world. This requires large amount of computational power, but gives much more information and is utilizable for a large range of problems, as compared to the second approach i.e. knowing only the information specific to the problem being solved. For example, specifically for obstacle avoidance task of an indoor ground robot, the robot may only need to know which regions of its way ahead are occupied by the obstacles. Information like the shape of objects, their absolute positioning in the world, or the understanding of the relationships between these objects may not be required for this specific problem. However, if we are talking of three dimensional (3D) space of a flying robot, all such information may be relevant, in order to utilize the third dimension of altitude and to join back the planned optimal trajectory safely, past the obstacle.

Typically, a vision system includes (but is not limited to) an image acquisition mechanism, followed by an image processor and an image segmentation system. This may be followed by some kind of image reasoning, which ultimately culminates into certain sort of scene modeling, comprising of desired attributes of the scene. Once the
desired scene model is obtained, decisions may be taken on how to achieve the vision system perceived goals (Figure 18).

Fig. 18. Overview of a Typical Computer Vision Problem. After successful solution of the problem, the vision based goals e.g. guidance and control, obstacle detection and avoidance, formation flight etc. should be achievable.

Successful motion through 3D space requires that any objects in the flight path must be avoided. This is not a prominent issue, once flight is at high altitudes. (Even birds are rarely seen above 8000 feet.) On the contrary, however, if the flight is close to the ground, e.g. within 1000 feet above ground level, obstacle avoidance is a very serious consideration. The various perceived roles for unmanned air vehicles fall within this flight altitude range. Such roles may include disaster management (e.g. in fires, earthquakes, floods, landslides, volcano eruptions, storms etc) and military or similar applications (e.g. reconnaissance, target identification, rendezvous, and nuclear,
chemical, biological and conventional warfare) [1]. Such perceived roles and tasks for UAVs require various capabilities like navigation and control, tracking, terrain mapping, formation flying, guidance etc., none of which is possible unless a collision free flight through the 3D space is ensured. For all such roles, the UAV essentially requires some kind of environment or scene sensing, which directly leads us to the requirement of vision-based systems.

In fact, vision systems are one of the most general sensors for robots and UAVs, since these deliver richer and more complete information than other sensors. For navigation in an unknown world, obstacle detection and avoidance is a fundamental behavior, which is a pre-requisite to build more complex navigational abilities. Hence the vision-based obstacle detection and avoidance directly contributes to a safe operation of a UAV.

There has been an extensive literature addressing obstacle detection and avoidance, particularly for ground robots. Various approaches to obstacle detection are roughly categorized here as follows.

**Multiple Sensor Based Obstacle Detection & Avoidance**

The most common approach to obstacle detection & avoidance is that of use of multiple sensors. Thus for example, David Coombs and Karen Roberts [37] propose two cameras looking obliquely to steer between objects. The left and right proximities have been compared to steer through the gap.

Another similar development is a vision system capable of guiding a robot through corridor-like environments by Argyros and Bergholm [38]. It uses three cameras, one for central forward vision and the other two for peripheral vision. The main principle is to implement a honey-bee-like reactive centering behavior by controlling the movement in a way that the optical flow on both sideward-looking cameras is equal. The normal flow for all three cameras is computed by an intensity-based algorithm, after
which, the depth to obstacles visible in the periphery cameras is extracted, by using the central camera to compensate for the rotational component of the ego-motion. The hardware requirements for this approach include three cameras and two workstations.

Analogous approaches have been proposed and successfully applied for various robotic platforms. Representative examples are: [39] for Stereo Vision (most common for ground robots), [40] for fusing radar and vision for obstacle avoidance on cars, and more significantly, [41] for Unmanned Aerial Vehicles flying through canyons.

**Single Sensor Based Approaches**

In his PhD Thesis [42] and relevant published work [43, 44], Randal C Nelson proposes the use of certain measures of flow field divergence as a qualitative cue for obstacle avoidance. It has been shown that directional divergence of the 2D motion field indicates the presence of obstacles in the visual field of an observer, undergoing generalized rotational and translational motion. Divergence information has been calculated from image sequences, based on the directional separation of optical flow components and the temporal accumulation of information. The use of the system to navigate between obstacles has been demonstrated by experimental results. This approach essentially does not do obstacle detection in 3D space, but instead comes up with a ‘No-Go’ direction, skipping directly to the obstacle avoidance part.

In their paper [12], Young et. Al. present an approach to obstacle detection, using optical flow without recovering range information. A linear relationship, plotted as a line called reference flow line, has been used to detect discrete obstacles above or below the reference terrain. The parameters of the reference flow line are estimated using the optical flow of a specific part of the picture that is assumed to be obstacle-free. Slopes of surface regions have also been computed. Objects that intersect with the reference space line and occlude it, cause different flow values than the reference line and can thus be detected. It may be seen that this approach may work effectively for ground robots in
general, and for UAVs during landing, but does not seem very useful in normal 3D flight of a robotic UAV, primarily because of absence of any reference or obstacle free terrain data in completely unknown flying environments.

Nicholas Hatsopoulos and James Anderson [15] also use optical flow, but instead calculate time to contact, which is an optical property. However, they describe in the paper that this approach, which has been proposed for collision avoidance in cars, is not effective in realistic driving environments, when the surfaces are not very flat and are not perpendicular to the center of camera axis.

**Other Three Dimensional Approaches**

Nakao et al [45] present a method of 3D shape reconstruction of objects for a camera mounted on a robotic arm with the object being modeled on the turn table. This approach effectively uses a single camera and an Extended Kalman Filter for 3D shape reconstruction. However, this paper does not seem to address the correspondence problem in detail, probably because there are very few feature points in the scene in such structured environment.

Besides, there had been a lot of literature under the heading of ‘Structure from Motion’ problem. Ref [46] analyzes many of such approaches.

In fact, one of the most relevant and significant contribution is that of Call and Beard [47] in which a ray intersection method has been used to identify 3D locations of obstacles for UAVs. The problem at hand may be considered as an improved solution approach to such a problem.

**Obstacle Detection & Avoidance in Structured Environment**

Ilic et al [48], present a monocular ground plane obstacle detection method using optical flow anomalies. The optical flow is computed on a single image row and compared to a model for ground point optical flow, obtained by direct calibration. This
approach seems efficient for ground robots but may not be suitable for UAVs, as the model for ground point optical flow may not be obtained for a completely unknown / unstructured 3D environment, which a UAV is expected to fly into. [13] and [14] also present approaches for obstacle detection and avoidance in either structured or partially known environments.

**Proposed 3D Reconstruction Approach**

The problem attempted in this research work is that of a single sensor, which is a camera and the solution being sought is that for a three dimensional problem in perfectly unknown world. Chapters 9 & 10 present the proposed solution based on full 3D Reconstruction or ‘Structures from Motion’ approach. The equations developed and the simulations results presented show that a 3D model of the scene can be generated from 2D image information from a single camera flying through a very small arc of lateral flight around the object, without the need of capturing images from all sides. The forward flight simulation results show that the depth extracted from forward motion is in fact usable for large part of the image, which is a significant contribution of this work. The approach is described with the potential test vehicle as GTMax UAV (Fig.19). A critical analysis of this and other approaches (Chapter 11) finally led to the use of Motion Estimation technique (Chapters 12 & 13).

A detailed description of the 3D Reconstruction approach follows.
Description of 3D Reconstruction Problem

3D Reconstruction or ‘Structure From Motion’ refers to the process of finding the three-dimensional structure of the environment using relative motion between robot/UAV and objects in the scene. When the observer moves, or the objects around him move, information is obtained from images sensed over time, which is then used to reconstruct the whole 3D space [49, 50].

The overall solution may involve solving various sub-problems as sensor calibration [51]; image acquisition; its processing and segmentation for desirable features; establishing correspondence between images, so that desired features could be tracked from one image to the next in time; and finally reconstructing the whole 3D space with objects in the scene and camera motion modeled [52]. This complete knowledge of the 3D environment is sufficient to enable a robot or a UAV, to successfully generate an obstacle avoidance maneuver [53, 54].

The problem is described under the following headings (also refer previous Fig.18).

Fig.19. GeorgiaTech GTMax UAV. Representative test vehicle for approach developed here. Equations developed in the subsequent chapter are based on this potential test vehicle.
Sensor Calibration & Image Acquisition

An image acquisition system generally comprises of one or more digital cameras. However, other sensors, aiming at specific attributes of the objects in the scene are also common. Examples include thermal imaging sensors, optical flow sensors, sonar, radar, laser range finders, infra-red sensors, ultraviolet sensors, etc. Camera or other sensors, in almost all cases require calibration (finding out intrinsic and extrinsic parameters for the sensor), before these could be effectively utilized for the problem at hand [51]. The potential test vehicle for the implementation of this work GTMax UAV (Fig. 19), is already equipped with the requisite system (hardware & software) [55], hence, this sub-problem has not been addressed in this work. The video images acquired on the GTMax are being digitized using a Frame Grabber.

Image Processing & Segmentation

The images from the frame grabber require lot of improvements. Hence the Image Processing aims at improving these images by attempting to reduce noise as far as possible, enhance contrast to a desired level, and even do data compression if required. This is followed by ‘Image Segmentation’, which extracts useful features from the output images of an image processor. Hence, various features points, edges, corners, surfaces, blobs etc. are identified and located, as a result of image segmentation. For this work, image processing and segmentation are considered the problems, which have already been addressed in earlier work [55, 56].

Correspondence Between Estimations & New Measurements

The next step after image segmentation is that of image reasoning, which involves collecting identified features into object shapes. Subjects of Pattern Recognition, size or motion recognition, feature tracking and feature correspondence, all can be viewed as various forms of image reasoning. Hence to establish correspondence between estimated
features and new measurements for features, Z-test [57] has been used in this work, as was proposed in [58]. Z-test is a statistical test used in inference to determine if the difference between a sample mean and the population mean is large enough to be statistically significant, that is, if it is unlikely to have occurred. In order for Z-test to be reliable, certain conditions must be met, most important of which, is that the population standard deviation must be known. Further, the sample must be a random sample, with a normal distribution of population sampling. In actuality, knowing the true standard deviation of a population is unrealistic (in which case a t-test must be used). However, in the case here, as the entire population of segmented feature points is known exactly in the image plane, Z-test is the preferred choice.

Noise and Non-Linearities

Noise in the image data is generally modeled as some random variations in brightness information. Such noise can originate in film grain, or in electronic noise in the input device (digital camera or other image acquisition media) sensor and circuitry, or in the unavoidable shot noise of an ideal photon detector [32]. For all the simulations in this dissertation, random Gaussian noise has been added so as to bring the simulations close to real world scenarios. Further, due to the non-linear system equations (see next chapter), Extended Kalman Filter has been chosen, which can treat this noise explicitly.

3D versus 2D

Solving a two dimensional problem of obstacle detection and avoidance, which is the case for most ground robots, is relatively simpler than solving a 3D problem. The 2D problem deals with the intensity map at each pixel on an image, by which, obstacles are identified that need to be avoided. Subsequent images indicate changes in the scene, which update this information for obstacle avoidance. The 2D obstacle detection hence generally solves only the problem of ‘directions to avoid’ and need not generate a 3D
scene model. For the specific case of obstacle avoidance, a ground robot may only need to know the regions of its way ahead that are occupied by obstacles. No information like shape of the objects, their absolute positioning in the world or the understanding of the relationships between these objects is required. Consequently the image data may be directly used without a reconstruction of the three-dimensional world of motion. Accordingly, no explicit knowledge about the camera parameters, ego-motion, and camera-to-ground coordinate transformations may be required [59]. On the contrary, a general 3D obstacle detection problem solves for all such attributes of world, and this is what has been addressed in this dissertation.

3D Modeling & Avoidance Maneuver

Once the problem of locating feature points in the 3D world is solved, the 3D coordinate information for all identified object features in the scene is obtained, from which a 3D model of the environment of the UAV could be generated. This is obviously far more computationally expensive, than a 2D case, but still is a preferred choice in this research work, due to the detailed information of the environment, that is obtained.

The knowledge of the 3D environment then enables an obstacle avoidance maneuver to be generated. This involves leaving the previous trajectory to avoid the unexpected obstacle, and then joining back the same trajectory in 3D space when past the obstacle, with minimum effort. This problem has been addressed in [53] and [54].
CHAPTER 9
PROPOSED 3D RECONSTRUCTION IN LATERAL MOTION [60]

Equations of Camera Motion

For the present problem, it is supposed that a camera is capturing 2D images and is mounted on a UAV. Immediately after the detection of feature points in the scene, UAV stops its forward flight and instead starts flying laterally around the object, following a circular path, where the flight path is tangent to the radial vector of the object. UAV flies in a radius of flight $r$, with angular velocity $\omega$ at a constant altitude $h$. The relative position of the camera in 3D space is $x, y, z$ and its orientation is $\phi, \theta$ and $\psi$ (Refer Fig. 20 below). This is an extreme case of obstacle avoidance maneuver selected to maximize predicted ability to generate the 3D map.

With the vehicle frame of reference as North-East-Down (NED), the following states and their rates are obtained for the camera

$$
\begin{bmatrix}
x
\end{bmatrix} = \begin{bmatrix}
rsin\omega t
r\cos\omega t
-h
\end{bmatrix} + \Delta Position
$$

and

$$
\begin{bmatrix}
x
y
z
\end{bmatrix} = \begin{bmatrix}
r\cos\omega t
-r\sin\omega t
0
\end{bmatrix} + \Delta Velocity
$$

(1)

where $x, y, z$ are the position states, with dot notation specifying the rate and $\Delta Position$ and $\Delta Velocity$ are the error values for position and velocity vectors modeled as Gaussian noise vectors of size 3x1, respectively. (Values of the noise covariances have been chosen keeping in view similar calculations e.g. in [53]). The orientation and orientation rates of the camera are given by

$$
\begin{bmatrix}
\phi \\
\theta \\
\psi
\end{bmatrix} = \begin{bmatrix}
\phi_c \\
\theta_0 \\
-\omega t
\end{bmatrix} + \Delta Orientation
$$

and

$$
\begin{bmatrix}
\phi \\
\theta \\
\psi
\end{bmatrix} = \begin{bmatrix}
0 \\
0 \\
-\omega
\end{bmatrix} + \Delta AngRate
$$

(2)
where $\phi$, $\theta$, $\psi$ define the orientation of the camera on the UAV, $\phi_c$ is the installation angle of camera on UAV, dot notation specifies the rate and $\Delta \text{Orientation}$ and $\Delta \text{AngVelocity}$ are the noise values for orientation and orientation rates modeled as Gaussian noise vectors of size 3x1, respectively.

![Diagram](image)

**Fig. 20. Camera Mounted on a UAV with a Detected Object in the Scene.** Lateral flight path after detection of an object, gives best 3D reconstruction.

For conversion between body frame and vehicle frame, the rotation matrix is as follows

$$L_{bv} = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \phi & \sin \phi \\
0 & -\sin \phi & \cos \phi
\end{bmatrix} \begin{bmatrix}
\cos \theta & 0 & -\sin \theta \\
0 & 1 & 0 \\
\sin \theta & 0 & \cos \theta
\end{bmatrix} \begin{bmatrix}
\cos \psi & \sin \psi & 0 \\
-\sin \psi & \cos \psi & 0 \\
0 & 0 & 1
\end{bmatrix}$$

(3)
and

$$L_{vb} = L_{bv}^T$$  \hfill (4)

**Z-test for Correspondence**

Statistical Z-test method has been used to solve the correspondence problem between the estimated corners from database and the measurements. The Z-test has been taken for a certain error index ($J$) and is the square of this index divided by its variance ($C$) i.e. Ztest value $= J^2/C$. Both the estimation error covariance (matrix $P$) and the measurement error covariance (matrix $R$) have been taken into account while calculating variance $C$ (Eqs. (5) and (6)). Then the Z-test value is inversely related to the likelihood of an event that a given measurement corresponds to the corner point chosen. Thus for example, if there is a large error between the measurement and the image data, but the measurement also has a large uncertainty, then the probability of its correspondence should be higher than the case in which, the measurement has a small uncertainty. Thus each corner point is to be assigned to a point, which attains the least Z-test value, meaning thereby, the highest likelihood.

For $Z$ being the projected measurement vector onto image plane and $X$ being the relative position vector in 3D space, the error index $J$ and its variance $C$ for calculating Z-test value are defined as

$$J = dx^2 + dy^2 \quad \quad \quad \quad \quad C = C_x P C_x^T + C_z R C_z^T$$  \hfill (5)

where

$$C_x = \frac{\partial J}{\partial X_v} \quad \quad \quad \quad \quad C_z = \frac{\partial J}{\partial Z}$$  \hfill (6)

are the two components of the variance $C$ of the error index $J$. 

60
In the Fig.21, \( Z \) is the projected measurement vector onto image plane and \( x_k \) is the projected database corner vector onto the image plane. Hence, it may be noted from Fig.21 that the residual vector is

\[
d = Z - x_k
\]  

(7)

Fig. 21. The Residual Vector on Image Plane. \( x_k \) is the projection of estimated feature onto image plane; \( Z \) is the new measurement; \( d \) is the residual vector between the estimated feature point and the new measurement with \( dx \) and \( dy \) as its components.

_**Pin-Hole Camera Model**_

Assuming that the camera is mounted at the center of gravity of the vehicle, let \( L_{bv} \) denote a known camera attitude represented by a rotation matrix from inertial to the camera frame. A camera frame is chosen so that the camera’s optical axis aligns with its \( X_c(t) \) axis. Then the relative position in camera frame will be as follows,

\[
X = X_v - X_p \quad \text{(in inertial frame)}
\]

(8)

\[
X_c = L_{bv} X \quad \text{(in camera frame)}
\]

(9)

where

\[
X_c = [X_c(t) \quad Y_c(t) \quad Z_c(t)]^T
\]

(10)

and the subscript \( v \) is used for vehicle position vector, the subscript \( p \) for the the object position vector, subscript \( c \) for the camera and \( X \) indicates a 3x1 relative position vector in 3D space.
Assuming a pin-hole camera model as shown in the Fig. 22, the object position in the image at a time step \( t_k \) is given by (\( x_k \) is a 2x1 vector in the image plane),

\[
x_k = \begin{bmatrix} y_k \\ z_k \end{bmatrix} = \frac{f}{X_{ck}} \begin{bmatrix} Y_{ck} \\ Z_{ck} \end{bmatrix}
\]  

(11)

This equation is non-linear with respect to the relative state. Hence an Extended Kalman Filter has been used here. In the implementation, focal length \( f \) of the camera has been assumed to be unity without loss of generality.

**Fig. 22. A Standard Pin-Hole Camera Model [54].** Coordinates of the object are known in the 2D image plane, but need to be found in the 3D camera or inertial frame.

Following Eqns 5, 6 and 7, expressions for the components of Variance matrix \( C \) as \( C_X \) and \( C_Z \), could now be written using chain rule as follows,
\[ C_X = \frac{\partial J}{\partial X_v} = \frac{\partial J}{\partial d} \cdot \frac{\partial d}{\partial x_k} \cdot \frac{\partial x_k}{\partial X_v} \quad \text{and} \quad C_Z = \frac{\partial J}{\partial Z} = \frac{\partial J}{\partial d} \cdot \frac{\partial d}{\partial Z} \quad (12) \]

**Extended Kalman Filter**

**Prediction Step**

The predicted stage (before the new measurement) of the Extended Kalman filter is defined by \( X_k^- = f(X_{k-1}^-, U_k) \), which for this case of no dynamics and no input for the feature point being modeled, simplifies to,

\[ X_k^- = X_{k-1}^- \quad (13) \]

The estimation covariance matrix is defined by \( P_k^- = F_k P_{k-1}^- F_k^T + Q_k \), which for no dynamics case simplifies to,

\[ P_k^- = P_{k-1}^- + Q_k \quad (14) \]

**Update Step**

The Kalman gain, the state and the estimation covariance for update stage (after the new measurement) are given by,

\[ K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} \quad (15) \]
\[ X_k = X_k^- + \sum K_k d \quad (16) \]
\[ P_k = P_k^- - \sum K_k H_k P_k^- \quad (17) \]

where \( R_k \) is the measurement error covariance matrix and \( H_k \) is the observation matrix Jacobian, which is defined as partial derivative of the residual \( (d) \) with respect to partial derivative of the state \( (X) \). It is calculated here as,

\[ H_k = \frac{\partial d}{\partial X} = \frac{\partial d}{\partial x_k} \cdot \frac{\partial x_k}{\partial X} \quad (18) \]
For the vectors $\mathbf{d}$, $\mathbf{X}$ and $\mathbf{x}_k$ defined above and $I_2$ as a 2x2 identity matrix, this is evaluated as (for detailed derivations please refer the Appendix A),

$$
\frac{\partial \mathbf{d}}{\partial \mathbf{x}_k} = -I_2 \quad \text{and} \quad \frac{\partial \mathbf{x}_k}{\partial \mathbf{X}} = \frac{1}{X_{ck}} \begin{bmatrix} -x_k & I_2 \\ \end{bmatrix}
$$

(19)

so that $H_k$ turns out to be

$$
H_k = - \frac{1}{X_{ck}} \cdot I_2 \cdot \begin{bmatrix} -x_k & I_2 \\ \end{bmatrix}
$$

(20)

Similarly the covariance matrix $R_k$ (measurement error covariance) is defined as,

$$
R_k = \frac{\partial \mathbf{d}}{\partial \mathbf{Z}} \cdot \mathbf{R} \cdot \left( \frac{\partial \mathbf{d}}{\partial \mathbf{Z}} \right)^T
$$

(21)

which is evaluated as (for detailed derivations please refer the Appendix A).

$$
R_k = I_2 \cdot \mathbf{R} \cdot I_2
$$

(22)

where $\mathbf{R}$ is the measurement noise covariance.

It may be seen now that the components of Variance matrix $C$, as $C_X$ and $C_Z$ given above, may be evaluated as (for detailed derivations please refer the Appendix A),

$$
C_X = \frac{\partial J}{\partial X} = \frac{\partial J}{\partial \mathbf{d}} \cdot H_k = 2 \mathbf{d}^T \cdot H_k
$$

(23)

$$
C_Z = \frac{\partial J}{\partial Z} = \frac{\partial J}{\partial \mathbf{d}} \cdot \frac{\partial \mathbf{d}}{\partial \mathbf{Z}} = 2 \mathbf{d}^T \cdot I_2
$$

(24)

**3D Modeling Algorithm**

The above equations have been implemented in the 3D scene modeling algorithm as shown in Fig.23. The algorithm starts when a feature point is detected in the scene. This information is fed to the program by the frame grabber after image processing and segmentation. Further, the camera calibration information is also being fed to the program by GTMax onboard systems, which includes its location in 3D space and installation angle on the UAV, besides the knowledge of its FOV (Field of View) and its
image plane size. The UAV then starts flying in a circular path of radius $r$. Eqs. (1) to (4) give the position, orientation and respective rates for the camera and Eqs. (8), (9) and (10) give the position information in camera and inertial frame. For the first iteration, as the estimation database is empty, all the feature points as measured in the image frame go into the estimation database without establishing any correspondence. Since this was only 2D information from the image plane, the third dimension is unknown and is supposed to be zero i.e. all points are supposed to lie on the ground plane initially. When the subsequent image information is received, the estimation points in the database are projected onto image plane (via Eq. (11)) and the residual vector is calculated between the new measurement and the estimated points on image plane (Eq. (7)).

Z-test correspondence is done to establish which measurement corresponds to which estimated value (Eqs (5), (6), (7) & (12)) and the new values are updated with the extended Kalman filter (Eqs. (13) to (24)). If correspondence is not established between a measured feature point in the image, with any of the estimated feature points, this feature point is recognized as a new point. Conversely if an estimated feature point existed, for which there was no corresponding measurement in the new image, this point is marked for deletion. However, it is actually not deleted unless it remains without correspondence for next consecutive N images. This has been done to ensure, that if a feature point temporarily goes out of view, it is not deleted immediately, otherwise the whole simulation time would increase, if it came back into the view later on and was instead recognized as a new feature requiring new estimation starting from the ground plane.
Fig. 23. The Proposed 3D Reconstruction Algorithm. The algorithm finds 3D coordinates of all feature points in the scene from 2D coordinates on the camera image plane, while UAV is flying around.
Simulation Results for Lateral Motion

As a first case, a cube was selected with eight corners (or eight feature points). This known model of the cube was used to verify the ability of the algorithm to successfully generate its 3D model using the 2D image information captured from a single camera. The simulation results are as presented in Fig.24. In this figure, the solid lines indicate the object to be modeled, the diamonds indicate the progressive outcome of corner estimation from the proposed algorithm, whereas the wavy arcs indicate the flight path of the camera with added Gaussian noise. The final plot at 60 sec of simulated flight shows that the diamonds approach the actual corners of the object being modeled, indicating a successful 3D obstacle detection for this case.

As a next case, a scene comprising of 35 feature points was chosen, as various corners of high-risers in a typical urban scenario. The simulation results for lateral flight path are shown in Fig.25. In this figure also, the solid lines indicate the object to be modeled, the diamonds indicate the progressive outcome of corner estimation from the proposed algorithm, whereas the wavy arcs indicate the flight path of the camera with added Gaussian noise. The final plot at 100 sec of simulated flight shows that the diamonds approach the actual corners of the object being modeled, indicating a successful 3D obstacle detection for this case as well. Table 3 below gives the values used for simulation.
Fig. 24. Lateral Flight Simulation Results with 8 Feature Points. Image processing is updated at 10 frames per sec. Convergence is good at 60 sec, traveling 25 deg around the object center, as the diamonds have approached the actual corners of the object.
Fig. 25. Lateral Flight Simulation Results with 35 Feature Points. Image processing is updated at 10 frames / sec. Convergence is good at 100 sec, traveling 40 deg around the object center, as the diamonds have approached the actual corners of the objects.
Table 3. Values Used for Lateral Flight Simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight altitude above ground level</td>
<td>140 ft</td>
</tr>
<tr>
<td>Radius of flight about the object</td>
<td>140 ft</td>
</tr>
<tr>
<td>Angular velocity around the object</td>
<td>0.36 deg/sec</td>
</tr>
<tr>
<td>Camera field of view</td>
<td>30 deg</td>
</tr>
<tr>
<td>Position error in all three states each</td>
<td>1%</td>
</tr>
<tr>
<td>Velocity error in all three states each</td>
<td>1%</td>
</tr>
<tr>
<td>Orientation error in all three states each</td>
<td>0.01%</td>
</tr>
<tr>
<td>Angular velocity error in all three states each</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

The simulation results of Figure 24 and 25 show that the proposed algorithm can successfully generate a 3D model of the scene, from 2D image information. This modeling only requires one camera as the sensor. The results have been achieved for an unknown world and no constraints were put on the environment being modeled. No attributes of the environments were provided to the system, except for the 2D images being captured by the camera. The scene modeling has been achieved (to within ±3% of actual 3D locations of the feature points) in 60 seconds of flight for 8 feature points, and 100 seconds of flight for 35 feature points. The successful 3D scene modeling required flying through a very small arc in lateral flight, as compared to the size of object being modeled. There had been no need to capture images from all sides of the objects being modeled.
CHAPTER 10
PROPOSED 3D RECONSTRUCTION IN FORWARD MOTION

Relevant Research

In their research paper [61], Matthies, Kanade and Szeliski present Kalman Filter-based Algorithms for Estimated Depth from Image Sequences. Besides other conclusions, they have shown that

1. For a translating camera, the accuracy of depth estimates increases with increasing distance of image features from the focus of expansion (FOE, which is a point in the image where camera translation vector pierces the image plane).
2. Best translations are parallel to the image plane and the worst are forward along the camera axis.
3. For practical fields of view, the accuracy of depth extracted from forward motion will be effectively unusable for a large part of the image. Thus for practical depth estimation, forward motion is effectively unusable compared with lateral motion.

Proposed Approach

The previous chapter demonstrated that lateral motion path gives good 3D scene modeling of objects from 2D image data. (In fact depth is just one coordinate of any feature point in 3D space). This substantiates deduction 2 above. This however, is apparently an awkward flight maneuver form a practical perspective in the sense that a UAV, which was supposed to fly forward, has to start flying laterally, as soon as some object is detected in the scene, in order to estimate its depth or 3D location in space. Hence, here an attempt was made to do depth estimation while flying forward, which is
in conflict to what was stated in conclusion 3 above. However, two facts are important here.

Firstly, estimation of 3D positions of those objects is attempted, which do not lie exactly at focus of expansion, because if the features exactly lie at FOE, there is no solution to the problem. This is in accordance with the first deduction mentioned above. It is proposed here that if the features are not at FOE, even flying forward could give reasonable depth estimation. Of course the accuracy would improve with increasing distance of features from FOE, as stated in [61] above.

Secondly, it may be noted that the conclusions in the above-referred paper were arrived at by linearizing the system equations and using a Kalman Filter. In this dissertation however, it is investigated, whether the use of non-linear Extended Kalman Filter instead of a regular Kalman Filter, can provide good results for forward motion of a UAV or a robot.

Accordingly it is proposed in this work that, subject to the two considerations just mentioned above, flying forward will give depth estimation, which is of practical use, as opposed to deduction 3 of above referred paper.

Implementation of this 3D obstacle detection in forward flight, changes only the equations of motion of camera i.e. Eqs. (1) for position and velocity of previous chapter. All other equations presented for Lateral flight in the previous chapter, remain valid in this forward flight case as well. This also applies to Fig.21 (Residual Vector), Fig.22 (Pin-hole Camera Model) and Fig.23 (Proposed Algorithm). The changes required in Eqs. (1) for position and velocity are: $\omega=0$ (for no lateral flight), 2nd (forward) component of position vector added with a factor $a \times t$, where $a$ is forward velocity and $t$ is time and second component of velocity vector added with this constant factor $a$. Hence the new equations are,
To avoid the obstacles at FOE, the speed of flight is a critical factor. If it is too high, the images of the objects, which are enlarging in this case as the motion is towards them, will quickly occupy almost whole of the image, including FOE as well. Hence the 3D scene modeling would not be possible. On the contrary, if the speed of flight is too low, there is less variation in the subsequent images, and hence less new information in those images for the Kalman filter update. This will in turn prolong the simulation time to an unacceptable extent. In this case of 35 feature points, the optimum speed of flight was found by iterations in repeated simulations, so as to achieve the correct 3D modeling at a relatively high speed.

**Forward Flight Simulation Results**

The simulation results for 3D obstacle detection in forward flight are presented in Fig.26. In this figure, the solid lines indicate the object to be modeled, the diamonds indicate the progressive outcome of corner estimation from the proposed algorithm, whereas the wavy line indicates the forward flight path of the camera with added Gaussian noise. The final plot at 125 sec of flight shows that the diamonds approach the actual corners of the object being modeled, indicating a successful 3D obstacle detection for this case. The table 4 below gives the values used for the simulation.
Fig.26 Forward Flight Simulation Results with 35 Feature Points. Image processing is updated at 10 frames/sec and UAV is flying forward at 1.4ft/sec. Convergence is good at 125 sec, as the diamonds have approached the actual corners of the objects.
Table 4. Values Used for Forward Flight Simulations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight altitude above ground level</td>
<td>140 ft</td>
</tr>
<tr>
<td>Forward Velocity</td>
<td>1.4 ft/sec</td>
</tr>
<tr>
<td>Camera field of view</td>
<td>30 deg</td>
</tr>
<tr>
<td>Position error in all three states each</td>
<td>1%</td>
</tr>
<tr>
<td>Velocity error in all three states each</td>
<td>1%</td>
</tr>
<tr>
<td>Orientation error in all three states each</td>
<td>0.01%</td>
</tr>
<tr>
<td>Angular velocity error in all three states each</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

The results presented in Fig.26 show that the proposed algorithm can successfully generate a 3D model of the scene, from 2D image information while flying forward towards the obstacle. The speed of flight is critical, as with too high a speed, obstacles will overlap the FOE. Too low a speed, on the contrary, will give very less new information for the update. Successful 3D modeling will not be possible in both such cases. Comparing the results of lateral flight simulations (previous chapter) and the results presented here, it can be said that, flight duration required to generate a 3D model of the scene while flying forward, was 25% more than the duration of flight required for lateral flight case. Subject to the two conditions of features not exactly at FOE and using EKF for non-linearities, the simulation results show that for practical fields of view, the depth extracted from forward motion is indeed usable for a large part of the image, which is exactly what was attempted here.
CHAPTER 11

ANALYSIS OF PROPOSED 3D RECONSTRUCTION APPROACH

The results presented in Chapters 9 and 10 for 3D scene reconstruction approach to object detection using images from a single camera, are critically analyzed here.

Computational Effort

For detecting $N_X$ number of feature points (in 3D space), $N_X \times N_Z$ correspondences need to be established, where $N_Z$ is the number of feature points picked up (observed) in every image. This holds for every iteration except the first one, when the database is empty and there is no correspondence to be done. So if the frame rate is $f$ frames per second and the simulation gives satisfactory results after $t$ seconds, then the total number of images used in the simulation are $N_I = f \times t$. Hence the total number of correspondences, the algorithm has to establish is given by $N_X \times N_Z \times N_I - 1$.

It may be realized that as the number of feature points $N_Z$ in the scene increases, the number of points in the database $N_X$ also increases accordingly (as the scene feature points eventually end up as points in the database, once the correspondence is established and 3D coordinates are found). Hence the computational effort increases tremendously, which in turn results in a need for much more simulation time requiring more and more images. For the simulations above, the computational effort increased by about 14.7 times with an increase in number of feature points by 4.4 times (precisely from 4.054 seconds required to simulate 8 feature points vis-à-vis 60.201 seconds required to detect 35 feature points to within $\pm 3\%$ of accuracy) for lateral flight. This further increased by yet another 25% for forward flight.
A real world scene may contain hundreds of feature points and the computational
effort required for such a scene could be quite intensive and may turn out to be
impracticable.

**Image Acquisition & Frame rate**

For the above analysis, a frame rate of 10 frames per second had been supposed.
Hence if the simulation ran for 60 seconds, 600 images were used (case of Fig.24 above).
With a better frame grabber / image processor, so that about 30 frames per second rate is
available (which is practical nowadays), the simulation time will reduce to 20 seconds,
which in turn means, lesser duration of lateral flight required, to correctly model the
scene.

**Error Analysis**

Fig.27 presents with error analysis, corresponding to figures 24, 25 & 26. One
sample point randomly has been chosen for each of the three cases. All the three
estimated coordinates of the selected feature points in the database have been compared
with the actual value of coordinates. Thus Fig.27 shows that all cases converge to the
actual point locations, to within about +3% indicating successful convergence of feature
points to their actual locations in 3D space.

In the algorithm however, simulation is stopped when the average error from each
of the three coordinates from all the feature points in the scene are successfully modeled
(to within about +3% of the actual locations in 3D space). This means that for a case of
35 feature points, there are 35 x 3 = 105 coordinates to be estimated correctly, before the
simulated flight ends.
Fig. 27. Error Percentage Versus Time for Sample Points. Plots a), b) and c) show estimation error progression for 60 sec in X, Y & Z coordinates respectively, for a sample point from Fig.24. Plots d), e) & f) are similar plots for a sample point from Fig.25, whereas plots g), h) & i) are for Fig.26 for 100 seconds of simulated flight.
Merits & Constraints of Proposed 3D Reconstruction Approach

It may be said that the obvious merit of this approach in both lateral and forward flight, is that of providing a capability of 3D obstacle detection and modeling by using only one camera. This is of significant importance for future miniature UAVs, which might not be capable of carrying any other sensor, except for a single camera. The information that is obtained as a result of this algorithm, is that of a full scale 3D model of the scene, which may be directly utilized for any mission planning, as desired.

On the contrary, the algorithm has an obvious constraint of tremendous increase in computational effort with an increase in number of feature points.

Further, the lateral flight pattern for such obstacle detection may also seem as a constraint, at least to a mission, which was that of moving forward towards the goal/target. The forward flight does overcome this constraint but increases computational effort by another 25%, and comes with an additional constraint of having no feature points at FOE.

Yet another constraint is that of at least having some feature points at all, in the scene. If the UAV takes-off e.g. in front of a flat wall, there are hardly any feature points to be detected and modeled, even with a lateral flight path. This constraint however is applicable to almost all vision based approaches. The only solution to such a problem is to supplement and integrate vision-based approaches with e.g. some laser range finders, sonar, radar etc. This obviously involves more sensors than one camera and being out of the scope has not been deliberated upon further, in this dissertation. Some reference to such work was though already presented in Chapter 8 (under Multiple Sensor Based Obstacle Detection and Avoidance).

Concluding 3D Reconstruction Approach

From the proposed algorithm and associated simulations it is concluded that,
1. The proposed algorithm can successfully generate a 3D model of the scene, from 2D image information.

2. This modeling only requires one camera as the sensor.

3. The results have been achieved for an unknown world and no constraints were put on the environment being modeled. No attributes of the environments were provided to the system, except for the 2D images being captured by the camera.

4. The 3D scene model gives information of size and location of all obstacles in the scene. This information is sufficient to initiate an obstacle avoidance maneuver in 3D space.

5. In the case of lateral flight, the scene modeling has been achieved (to within $\pm 3\%$ of actual 3D locations of the feature points) in 60 seconds of simulated flight for 8 feature points, and 100 seconds of simulated flight for 35 feature points.

6. The successful 3D scene modeling required flying through a very small arc of lateral flight as compared to the size of object being modeled. There had been no need to capture images from all sides of the objects being modeled. Thus the approach is much better as compared to a typical ‘Structure from Motion’ problem, which may require right, left, top or other views of the object, in order to generate its 3D model.

7. In the case of forward flight, the speed of flight is critical, as with too high a speed, obstacles will overlap the FOE. Too low a speed, on the contrary, will give very less new information for the update. Successful 3D modeling will not be possible in both such cases, while flying forward.

8. Comparing the results of lateral flight simulations (Chapter 9) with that of forward flight (Chapter 10), it can be said that, flight duration required to generate a 3D model of the scene while flying forward, was 25% more than the duration of flight required for lateral flight case.
9. Subject to the two conditions of features not exactly at FOE and using EKF for non-linearities, the simulation results for forward flight show that the depth extracted from forward motion is indeed usable for a large part of the image.

10. The algorithm does require some feature points in the scene, both in the case of lateral flight as well as the forward flight. Hence if no feature points are detected in the scene, the algorithm implies that there are no obstacles to be avoided and the initial flight path of the UAV may be continued without any disruption. This is almost always true in real world scenarios.
Further to the analysis presented in the previous chapter, it may be noted that employing any of the two types of camera translations, whether parallel to the image plane (Chapter 9), or perpendicular to it (Chapter 10), comes with some inherent practical limitations. Regarding first,

1. Computational effort increases tremendously with the increase in number of feature points in the scene. This was primarily due to the effort involved in solving for the correspondence problem. As a real world scene may comprise of hundreds and hundreds of feature points, the 3D reconstruction or structure from motion approach may become quite impracticable.

2. A motion parallel to image plane, though gives best camera translations for 3D reconstruction [61], it is a disruption to the normal path of the UAV/robot, which was earlier moving forward towards its destination. The robot/UAV had to then interrupt its forward motion, as soon as some features were detected on its image plane, and had to generate a 3D model, in order to ensure its safe and collision free path ahead.

3. Many UAVs or ground robots may not be capable of executing such a lateral motion while keeping the camera looking forward. In general, rotorcraft can execute such a maneuver, but fixed wing aircraft cannot. Hence the 3D reconstruction / structure from motion approach would be pretty inadequate for all such robots or UAVs, incapable of executing such a lateral motion/maneuver.
That is why Chapter 10 proposed perpendicular translations to camera plane (forward motion if camera was looking ahead), which would neither be a disruption to normal forward motion of a UAV / robot, nor would be dependent on its lateral motion capability. However, the first constraint mentioned above of tremendous increase in computational effort with the increase in number of feature points, does apply equally to this approach as well. Further, this approach comes with another stringent constraint. This is to ensure that obstacles do not overlap the focus of expansion during the course of 3D reconstruction process, because depth estimation at FOE is not possible [61]. To implement this, an optimum forward velocity is to be known a priori. This optimum velocity is to guarantee that the motion is neither too slow, so that enough information is available for the update; and nor it is too fast, so that the obstacles in the scene may not quickly enlarge and occlude the focus of expansion. Though such a velocity was found out by repeated iterations in Chapter 10, in reality, knowing such an optimum velocity a priori, may turn out to be unfeasible. Hence some other approaches are considered next.

**Optical Flows**

Another effective approach available in the literature to solve the problem of object detection using a single camera for collision avoidance is that of use of Optical Flows. Optical flow is defined as the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (eye/camera) and the scene [62].

Although most researchers have used optical flows with multiple cameras on robots/UAVs, quite a few have used this technique to solve the problem of object detection for collision avoidance using a single camera [12-16]. Of these, the most relevant contribution is that of [16], in which it has been shown that,

1. Optical flows could be used for obstacle avoidance by turning away from the regions of high image velocity.
2. One can generate a forward collision response by measuring relative rate of expansion.

3. Centering response between obstacles could be generated by equalizing optical flows on left and right side in an image sequence.

It may be realized that, this approach seems to overcome all the constraints of 3D Reconstruction or Structures From Motion approach. Thus the calculations of optical flow vectors are independent of number of feature points in the scene and no disruption in the normal forward motion is required, which makes this approach independent of lateral motion maneuver capability.

This approach however comes with its own inherent constraints i.e.

1. Typical solution to the Optical flow problem involves intensity constraint equation and smoothness constraint equations [10]. Hence the solution necessitates that intensity remains constant from one image to the next in time and there exists smooth motion of pixels in these images. Ensuring that light intensity remains constant for the robot/UAV environments may be impractical in many situations.

2. The approach is quite computationally heavy as the typical solution involves calculating an optical flow vector for every pixel in the image. With image sizes growing tremendously with the modern day technology, flow vector calculations for every pixel of every image in the video stream may not suit a real time application.

3. The approach is also very sensitive to noise. Almost all video data comes with noisy pixels, and calculations of flow vectors for each one of such pixels, may waste a lot of computational resources, besides generating useless flow information.
4. Needless to say that own-ship motion correction does need to be applied to such vectors, before these could be utilized effectively for obstacle detection and avoidance.

5. The approach is also very sensitive to camera shake or vibrations. Such shake or vibrations would generate an undesirable optical flow vector field. Almost all UAVs/robots, whether due to own ship motion or due to motion of their rotating parts (propellers, motors, actuators etc.) will impart shake or vibrations to the camera, leading to undesirable optical flow vectors.

Flow Field Divergence

Another technique that was proposed in [43-45] was to use certain measures of flow field divergence as a qualitative cue for obstacle avoidance. This research showed that directional divergence of the 2D motion field indicates the presence of obstacles in the visual field of an observer, undergoing generalized rotational and translational motion. Divergence information was calculated from image sequences, based on the directional separation of optical flow components and the temporal accumulation of information. The use of the system to navigate between obstacles was demonstrated by experimental results.

Although one could draw similarities in this approach with the optical flow approach, the mathematical equations developed and implemented, were quite different than the standard solution [10] of an optical flow problem.

Like optical flows, this approach is also independent of number of feature points in the scene. The approach was implemented using averaging filters as well, so it would be less expensive than optical flow vector calculation, (which is typically done for every pixel). Its sensitivity to noise and camera shake would also be reduced as a result of such filters. Its application however, was only demonstrated for objects persistently present in the scene as it may not apply to objects entering or leaving the scene. Hence the
approach could be regarded as better than general optical flow technique in some sense, but may be less capable than optical flows in other areas.

**Proposed Motion Estimation**

Taking into account the exploration and analysis of existing methodologies as discussed above, use of motion estimation is proposed here to solve the problem of object detection for collision avoidance using a single camera.

Motion estimation is the process of determining motion vectors that describe the transformation from one 2D image to another in time. The motion vectors may relate to the whole image (global motion estimation) or specific parts, such as rectangular blocks, arbitrary shaped patches or even per pixel. Like flow field divergence, this concept is also closely related to optical flow concept, however the most significant difference is that ‘exact’ correspondence of pixel positions as in optical flows is not a must with motion estimation. Rather than searching for an ‘exact’ match, this approach searches for the ‘best’ match. This fact alone makes the underlying mathematics of this approach, much more robust as compared to optical flows.

The motion estimation technique is widely used in video compression standards e.g MPEG-1 to MPEG-4, [20]. The most common algorithm to do motion estimation is that of Block Matching Algorithm (BMA) in all standard video codecs [21]. An overall block diagram of motion estimation technique is presented in Fig. 28.
Figure 28. Overall Block Diagram of Typical Motion Estimation. Motion Vector field is generated between images at time $t+1$ and $t$. This vector field could then be utilized for various tasks e.g. video compression (most common), image stabilization, motion prediction and as proposed in this work for collision avoidance.

As this approach is based on blocks of pixels rather than individual pixels, hence the computational cost is much lower as compared to Optical Flows. A block of pixels also acts like an inherent noise smoothing filter for the whole block. Hence the approach is far less noise sensitive than single pixel based optical flows. By the same argument, it will be much less sensitive to undesirable flow vectors from camera vibrations and shake. The block based approach is also much less sensitive to individual pixel intensity changes and abrupt motions. Further, all the three relevant merits and capabilities of optical flows approach, as mentioned above referring [16], are retained in a BMA.

A decision matrix based on Multi Criteria Decision Analysis (MCDA), [63, 64] against all the criteria discussed so far, is presented below. It is seen from this analysis that Motion Estimation technique meets all criteria discussed.
Table 5. Decision Matrix Based on Multi Criteria Decision Analysis

Problem:
Object detection using images from a single camera for collision avoidance

<table>
<thead>
<tr>
<th>Alternatives</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Lateral Flight 3D Reconstruction</td>
<td>0</td>
</tr>
<tr>
<td>Forward Flight 3D Reconstruction</td>
<td>0</td>
</tr>
<tr>
<td>Optical Flows</td>
<td>1</td>
</tr>
<tr>
<td>Flow Field Divergence</td>
<td>1</td>
</tr>
<tr>
<td>Motion Estimation</td>
<td>1</td>
</tr>
</tbody>
</table>

Legend:
- 0 indicates unacceptable or no capability
- 1 indicates desirable capability
- Highest total in the last column indicates the best alternative

Criteria (desirable characteristics of an object detection/avoidance algorithm):

A. Independent of external knowledge source for aircraft states
B. Obstacle detection without disruption of current flight path
C. Very little or no constraint on speed of flight
D. Valid for both fixed wing and rotary wing aircraft
E. Real-time implementable computational effort
F. Computational effort independent of number of features in the scene
G. Independent of intensity changes in the scene
H. Independent of object motion smoothness constraint
I. Quickly distinguishes closer objects from far off ones (scene depth sensitivity)
J. Good noise handling
K. Detection of new obstacles entering the scene
**Block Matching Algorithm (BMA)**

In a typical BMA, a frame is divided into blocks of $M \times N$ pixels, or more usually, square blocks of $N^2$ pixels [21]. Then, for a maximum motion displacement of $p$ pixels per frame, the current block of pixels is matched against a corresponding block at the same coordinates but in the previous frame, within the square window of width $N+2p$ (Fig. 29). The best match on the basis of a matching criterion yields the displacement.

Various measures such as the cross-correlation function (CCF), mean squared error (MSE) and mean absolute error (MAE) can be used as the matching criterion. For the best match, in the CCF the correlation has to be maximized, whereas in the latter two, the distortion must be minimized. In practical coders both MSE and MAE are used, since it is believed that CCF would not give good motion tracking, especially when the displacement is not large [21]. The matching functions of the type MSE and MAE are defined as:

For MSE:

$$M(i,j) = \left[ \sum_{m=1}^{N} \sum_{n=1}^{N} \{ f(m,n) - g(m+i, n+j) \}^2 \right] / N^2, \ -p \leq i, j \leq p \quad (25)$$

and for MAE:

$$M(i,j) = \left[ \sum_{m=1}^{N} \sum_{n=1}^{N} | f(m,n) - g(m+i, n+j) | \right] / N^2, \ -p \leq i, j \leq p \quad (26)$$
Figure 29. **Block Matching Algorithm.** The current and previous frames in a search window of dimension \((N+2p)\times(N+2p)\) is shown. The current frame block is shaded gray and its closest match is to be found in the search window in the previous frame based on a chosen matching criterion e.g. CCF, MSE or MAE [21].

**Preliminary Implementation: Motion Estimation**

Implementation of motion estimation based on block matching algorithm requires a matching block search methodology, which is discussed below. Results from preliminary implementation on synthetically generated images as well as real images, follow.

**Exhaustive Search Block Matching**

There are quite a few fast block matching search algorithms available in the current and past research. Examples are ‘Simple and Efficient Search’ [23], ‘Four Step Search’ [24], ‘Diamond Search’ [25], ‘Adaptive Rood Pattern Search’ [26] etc. All of these approaches though make the algorithm quite fast in terms of computational cost, but do come with some inherent data loss. It was aimed here to avoid any data loss.
whatsoever; hence an ‘Exhaustive Search’ (or Full Search) approach is used in the work presented here on some synthetically generated images first. Although Adaptive Rood Pattern search was also implemented and tested against the same images and it turned out to be about 3-4 times faster than the Exhaustive search, but gave slightly different obstacle avoidance maps than Exhaustive search, which made it less reliable than exhaustive search. Hence Exhaustive search was the preferred choice.

**Implementation on Synthetically Generated Images**

Exhaustive search block matching algorithm using raster scan pattern was implemented and results from a simple Matlab implementation some synthetically generated images, are presented in Figure 30. The images were generated with the background fixed and a supposed obstacle entering the scene from left top corner. Motion vectors were calculated and overlaid (in red) upon the current frame, pointing from the current frame to the previous one. It may be seen that the obstacle entering the scene could be clearly identified from the motion vectors generated in that area of the image only. Rest of the scene though contains hundreds of feature points, but no motion vectors are generated from those feature points. Hence the identification of the obstacle is not related to the background information and its complexity with regards to number of features already in the scene.

Similar results could be obtained for a case in which an obstacle is moving relatively towards own-ship, and will be enlarging in the scene with its edges in motion. Such a motion will generate similar motion vectors as of Fig. 30 and hence would clearly identify the obstacle to be avoided. Yet another similar scenario would be that of many obstacles in the scene moving relative to own-ship. The closest obstacles then, would generate maximum motion vector magnitudes, which would increase at a much faster rate from one frame to another in time, as compared to far off obstacles. Hence we would be
able to identify the most critical or closest obstacles that need to be avoided in the scene, not only from large magnitudes, but also from the rate of change of these magnitudes.

Figure 30. **Exhaustive Search BMA Implementation on Synthetically Generated Images.** An obstacle enters the scene from second image onwards. It is easily identifiable from the presence of motion vectors in that area of the scene only. (Motion vectors are calculated from the new image to the old one and are overlaid on the new image).

**Implementation on Real Images**

Next, some videos were recorded by a camera installed on Georgia Tech’s GTMax UAV [55], with a calibration board attached to a rope, entering and moving in the scene and the aircraft not in flight. This was very close to hardware in the loop tests. The BMA implementation correctly identified the obstacle along with its rope entering the scene, by the presence of motion vectors there. However, some noisy & undesirable vectors are also seen, especially in the sky area (Fig. 31).
Figure 31. **Exhaustive Search BMA Implementation on Real Images from a UAV on Ground.** An obstacle (calibration board attached to a white rope) is moving in the scene. Motion vectors appearing there, correctly identify the board and its rope. However, incorrect/noisy vectors are being generated in the almost uniform colored sky.

As a next stage, a video was recorded from a follower UAV (GT Yak) in a formation flight with another UAV (GT Edge) as the leader aircraft. A video clip was considered in which the leader aircraft is reducing altitude, which should be identified as the only potential obstacle to be avoided in the scene, as everything else is sufficiently far away and should not generate large motion vectors necessitating a collision avoidance maneuver. Fig. 32 presents results of BMA implementation on such a video clip.
Figure 32. Exhaustive Search BMA Implementation on Real Images from a UAV in Flight. The leader aircraft in the scene is reducing altitude in the scene. A correct BMA implementation should have given large motion vectors only at the location of leader aircraft. However, quite a few undesirable motion vectors also appear elsewhere in the scene, making it difficult to identify the only potential obstacle as the leader aircraft, requiring a collision avoidance maneuver.

Desirable results from this BMA implementation should have given large motion vectors at the location of the leader aircraft changing altitude in the scene. However, it is observed in Fig. 32 that quite a few undesirable motion vectors appear, making it rather difficult to identify clearly the leader aircraft as a potential obstacle to be avoided. These implementation issues are investigated next and various solutions are proposed in the following chapter.
CHAPTER 13

HANDLING UNDESIRABLE & NOISY VECTORS FROM MOTION ESTIMATION IMPLEMENTATION

Probable Causes

The undesirable vectors in the Figures 31 and 32 could have originated due to many reasons. Following are identified as four main probable causes,

1. Uniformity in color.
2. Camera vibrations & shake.
3. Own-ship or ego motion.
4. Image noise.

It may be noted that the BMA approach is based on finding the best matching block with minimum error in the search window as per the matching criteria (Equations 25 and 26, in the previous chapter). Hence if all blocks in the search window are almost of same color, there would be minimal variation in the error and the correct minima would be hard to find. In many cases, with so less variation in the error between blocks of pixels, an undesirable match would occur. Hence large blobs of uniform color would lead to undesirable motion vectors. This was a typical case of Figure 31 above, where almost all undesirable vectors appeared in the sky. A simple texture filter is proposed to resolve such a problem. It is discussed below.

Proposed Texture Filter

Specifically referring to Fig. 31 above, almost all the undesirable vectors are originated in the sky area, which is uniform in color. Although there are quite a few techniques for filtering textures, as summarized in [65], a very simple and low cost
threshold filter is proposed here, which gave quite satisfactory results in our application. This proposed filter, involves calculating the variance in intensity of every block in the target image frame. If that variance falls below the minimum threshold variance, than the motion vector for that block is suppressed. The threshold variance was established experimentally by calculating variances of uniform colored blocks in various test images. The BMA algorithm is added with this texture filter as shown below (Fig. 33).

**Figure 33. BMA with Texture Filter.** The texture filter (shaded blocks above) was added to Block Matching Algorithm in order to handle undesirable motion vectors generated at uniform colored blocks in the images.
Testing this updated algorithm against images of Fig 31, the resulting vectors obtained are shown in Fig. 34.

Figure 34. BMA with Texture Filter Applied to Real Images from a UAV on Ground. The texture filter along with BMA was applied to real images of Fig. 31 above. Almost all undesirable vectors in the sky are eliminated, with only the desirable vectors remaining. These remaining vectors indicate potential obstacles to be avoided in the scene i.e. an obstacle entering the scene.

It may be seen from Fig. 34 above that addition of the proposed threshold based texture filter to a BMA implementation has successfully eliminated almost all undesirable vectors in the uniform colored sky. The vectors that remain clearly identify the only moving obstacles in the scene. Hence the efficacy of the proposed algorithm to identify
potential obstacles in the scene is demonstrated for these images. It may be noted however, that these images were taken from a camera mounted on a UAV (Georgia Tech’s UAVRF’s GTMax vehicle [55]), while it was on ground.

The same algorithm incorporating the proposed threshold filter, was next applied to real videos of Fig 32 taken from a UAV in flight. The resulting vectors are shown in Fig. 35.

---

**Figure 35. BMA with Texture Filter Applied to Real Images from a UAV in Flight.** The texture filter along with BMA was applied to real images of Fig. 32 above. Quite a few undesirable vectors in the sky are eliminated. The remaining vectors are still insufficient to identify the leader aircraft in the scene. Hence the texture filter alone does not suffice all real world scenarios.
Comparing Figs 32 and 35, it is observed that although many undesirable vectors in the sky area have been eliminated, there are quite a few undesirable vectors around the horizon and in the ground area, that remain. These vectors hence do not clearly identify the only potential obstacle (leader aircraft) in the scene. It is therefore, concluded that just a texture filter with BMA is not sufficient to handle all real world scenario.

Next, the reasons for these undesirable vectors are further investigated. Of the reasons listed in ‘Probable Causes’ para above, the most obvious reason is that of camera vibrations and shake. This is understandable because this in fact, is one of the main differences between the images of figures 34 and 35. Indeed camera vibrations could be generated because of propellers, tail rotors, actuator motors, or other moving parts, as well as own-ship motion through air, whether for a flying UAV, or for a ground robot in motion. Hence adequate solutions need to be sought in order to remove undesirable vectors appearing in Fig 35, due to either camera vibrations & shake, ego-motion, image noise or other causes.

**Camera Vibration**

Since the camera vibration do induce undesirable motion vectors, two categories of methods to handle this problem are proposed here. These are,

1. Image Stabilization (IS)
2. Vibration Damping.

**Image Stabilization (IS)**

Image Stabilization is a family of techniques used to reduce blurring associated with the motion of a camera or its subject. It is commonly used in binoculars, still and video cameras and astronomical telescopes. It is important to note that IS cannot prevent motion blur caused by the extreme movements of the camera. It is only capable of reducing minute shake and vibrations, specially associated with human hand in hand-held
cameras. Larger motions must be corrected otherwise (This is handled via ego motion correction and image noise handling techniques as discussed below)

There are quite a few implementation techniques of image stabilization [66-70]. These include,

1. Lens-Based Optical Stabilization,
2. Sensor-Shift Optical Image Stabilization,
3. Digital Image Stabilization,
4. Stabilization Filters,
5. Orthogonal Transfer CCD,
6. Camera Body Stabilization, etc.

Depending on various pros and cons of these techniques, it is proposed to use here both Optical IS (most effective for low frequency vibrations) as well as Digital IS (for higher frequency vibrations). Unfortunately most of the video cameras available in the market only use optical IS, because these are most effective against human hand-shake like vibrations. There are very few video cameras which do use Digital IS. However, (to the best of author’s belief and knowledge) there are no video cameras available which incorporate both Optical as well as Digital IS, which is in fact the preferred choice here. It may be said that technology in the near future may come up with such video cameras. Meanwhile, it is proposed here to use the Digital IS video cameras available (e.g. Sony’s FCB-EX980S or Hitachi’s VK-S654 cameras) for implementation of this work.

**Vibration Damping**

The second category of methods, which is proposed here to handle camera vibrations, is that of use of vibration damping. This includes using vibration insulators and shock absorber mounts for the camera installation. This obviously will help in all kinds of vibrations, whether low frequency (similar to human hand shake), or those typically induced by a propeller, tail rotor, actuator motors and other moving parts in a
flying UAV or a ground robot in motion. Appendix C describes a method to choose vibration isolators in general, and as applied specifically to our implementation.

**Ego Motion Correction**

Ego motion or own-ship motion relative to the scene, would generate a motion field with vectors in the scene, that are not required to be used for obstacle detection for avoidance. Hence we need to correct for this motion to take out those own-ship motion induced vectors.

An important consideration is to be noted here. The motion of our own aircraft is in 3D space, whereas the motion vectors generated from BMA are 2D, with apparently the depth dimension lacking. Hence, especially in regards to forward motion, a linear subtraction of forward velocity vector is not possible from the 2D motion vectors, which were generated in vertical-lateral image plane. A normalization approach instead is preferred over linear subtraction and hence has been used here. (Further details are presented in the ‘Implementation of all Proposed Solutions’ para in the end).

**Image Noise and its Handling**

Besides various factors contributing to undesirable motion vectors in real-world scenarios as discussed above, image noise is yet another significant factor to generate undesirable motion vectors. Such noise (referring discussions in Chapter 6 & [32]) could come from

1. Film grain noise (depending on type of camera).
2. Interference with other onboard electronic/avionics equipment.
3. Interference with the wireless link.
4. Digital video camera and circuitry noise.
Two methods are proposed here to handle such noise. These are: Temporal Accumulation and Spatial Filtering.

**Noise Filtering by Temporal Accumulation**

Temporal Accumulation is the addition of subsequent motion fields in time [42-44]. This technique has been tested and tried in the work presented here on various video clips and has been found to be an effective tool to remove unwanted noisy motion vectors. In specific, this approach works three-ways,

1. Supposing that all noise generated in the images is random, leading to randomly noisy vectors, subsequent motion vector fields generated in time, once added together, would cancel the randomly directed vectors.

2. It enhances the desirable motion vectors. If the motion of potential obstacles in the scene is not very random, and stays in one direction, at least for a few frames (which amounts to just a fraction of a second, and is therefore a very practical assumption for almost all real world objects), then adding the subsequent motion vector fields will add up and enhance the desirable vectors of that object in the scene. This would result in large vectors, easily indicating the presence of the potential obstacles to be avoided in the scene.

3. Temporal accumulation also alleviates to an extent, the camera vibrations and shake problem, if vibrations were truly random. Hence this technique not only in itself, is an effective method to deal with unwanted motion vectors problem, but also substantiates other techniques proposed here.

**Noise Handling by Spatial Filtering**

Spatial Filtering is an established method to alleviate image noise [32]. It must be noted however, that all noise filters come with some information loss. Out of various options that were tested and tried to solve the problem here, including linear noise filters
(e.g. Averaging filter, Gaussian filter, Laplacian filter, Difference of Gaussian filter, Laplacian of Gaussian filter etc.) as well as non-linear filters (e.g Rank filter, Median filter etc.), we did find the median filter as the most suitable one for our application. However, as mentioned, it did come with a data loss, especially when the obstacles in the scene were small enough, to be of the order of a block size. Since use of any of such filters would have been an additional constraint on the problem, none of these filters was chosen for the real-time implementation as proposed here.

Instead, a simple spatial filter was proposed and used here, which was proven to come with almost no loss of pertinent information. It was based on $L_2$ norms of motion vector magnitudes and was incorporated with the Ego-motion corrector as described below.

**Implementation of all Proposed Solutions**

For handling undesirable and noisy vectors generated from uniform colored blobs, texture filter implementation has already been discussed above.

To deal with camera vibrations and shake via image stabilization, selection of a suitable camera with in-built image stabilization has also been discussed above. A summary of specifications of the chosen camera is presented in Appendix B, whereby the most significant feature is that of ‘Image Stabilization’.

Suitable vibration insulators were selected off-the-shelf keeping in view the forcing frequencies, weight and size limitations of the implementation platform. The complete procedure of selection of appropriate vibration insulators is summarized in Appendix C.

The implementation steps for ego motion correction and image noise handling via temporal accumulation and spatial filtering are discussed next (Overall block diagram is presented in Fig 39 and the implemented code in Visual C++ is presented as Appendix E).
1. First, for every two consecutive images in the video sequence, the motion vector field is calculated from Block Matching Algorithm as described in the previous chapter (see also Figs 31-33). The texture filter is inherently imbedded into this algorithm as discussed under Proposed Texture Filter and Figures 33-35 above. Hence all the undesirable or erroneous vectors generated from large uniform colored blobs like sky, are eliminated at this stage.

2. Similarly, sixteen such consecutive motion vector fields are generated from seventeen images in time. Thus first vector field is generated between first and second image, second vector field is generated between second and third image and so on, until sixteenth vector field is generated between sixteenth and seventeenth image.

3. Temporal accumulation is done now by adding all the sixteen motion vector fields generated in time. Any random noisy vectors present in these fields would thus be eliminated by this temporal accumulation.

4. For spatial filtering and own-ship motion correction, vector magnitudes of all the vectors are calculated simultaneously. Thus for a vector field, where the image size was MxN pixels and a square block size of ‘mbSize’ was used in the BMA, then we have MxN/mbSize² vector magnitudes for this vector field. As an example case, if image size in pixels was 320x240 and the block size used was 8x8, then we have 320x240/8² = 1200 vector magnitudes for this vector field.

5. In each vector field, these 1200 vector magnitudes are separated into 12 square matrix sections of 10x10 magnitudes each. The matrix sections are named 11, 12, 13, 14 (first row), 21, 22, 23, 24 (second row) and 31, 32, 33, 34 (third row). These matrix sections are shown overlaid on a typical vector field (after texture filter) in Figs 36 and 37 (also refer Fig 34).

6. Supposing that camera is looking straight ahead, the matrix sections 22 and 23 are exactly ahead of us and any obstacles detected exactly in front are obviously
more critical for collision avoidance than all outer matrix sections 11, 12, 13, 14, 21, 24, 31, 32, 33, 34. This is because in many cases, depending on camera field of view, collision avoidance may not even be required for these outer matrix sections as we may fly under, over or from sides of these matrix sections. Furthermore, the outer matrix sections being farther from focus of expansions (location of focus of expansion for own-ship motion correction is marked in Fig 36 for camera looking straight ahead), would generate larger motion vectors than the straight ahead sections 22 and 23. This is because the objects at the sides appear to move faster in the scene than the objects directly in front. For both of these reasons, in the code implementation, a weightage factor greater than one is applied to the vector magnitudes from straight ahead matrix sections 22 & 23, as compared to the outer matrix sections.

7. At this stage spatial filtering is done by utilizing matrix norms. Holder norms for matrices are defined by choosing \( \| \cdot \| = \| \cdot \|_p \) [71]. Hence for matrix \( A \in \mathbb{R}^{n \times m} \),

\[
\| A \|_p \equiv \left[ \sum_{i=1}^{n} \sum_{j=1}^{m} | A(i,j) |^p \right]^{1/p}, \quad 1 \leq p < \infty
\]

\[
\text{max} \quad | A(i,j) |, \quad p=\infty
\]

Here, since all matrix sections are 10x10 size, hence \( m=n=10 \). Further, \( p=2 \) is chosen for \( L_2 \) norm. The norms for matrix sections 22, 23 are weighted 1.25 times more than their actual values, due to the reasons described in para 6 above.

8. Further to para 6, supposing sufficient texture in the scene (which is a requirement for almost all vision based systems), the section matrix norms are normalized against the minimum motion observed in the outer matrix sections. This minima actually indicates own motion in the scene, as all motion more than this minima is
actually coming from obstacles in the scene. Hence this normalization leads to correction for own ship motion.

9. A forward collision avoidance map is thus generated, with highest numbers indicating maximum probability of obstacles ahead and minimum numbers indicating suggested direction to turn to, to avoid obstacles. (Fig 38)

---

**Fig. 36. Sections of Vector Field Overlaid on an Image Showing all Motion Vectors After Texture Filtering.** The image is 320x240 pixels, with each vector representing motion of an 8x8 pixel block from the previous image. Hence there are 320x240/8^2 = 1200 vectors generated, whereby the non-zero vectors are shown above. The corresponding vector magnitudes are separated into 12 sections where each section has 10x10 vector magnitudes. The position of focus of expansion is marked for a camera looking directly in front and moving straight ahead. This is required for ego-motion correction.
Fig 37. Motion Vector Magnitudes Used for Spatial Filtering and Own Ship Motion Correction. Motion vector magnitudes are divided into 12 matrix sections with the central forward matrix sections being more critical than all the outer sections. Forward central matrix sections not only indicate obstacles directly on forward collision path ahead, but also generate lesser motion, being closer to focus of expansion. Hence a weightage factor of >1 is applied to these matrix sections.

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>12</th>
<th>8</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Fig 38. A Sample Forward Collision Avoidance Map. Highest numbers indicate maximum probability of obstacles ahead, whereas, minimum numbers indicate suggested directions to turn to, to avoid collision with obstacles ahead. The forward central sections (with values 1 & 0 here) being directly on collision path ahead, were generated by applying a weightage factor greater than one. The values above correspond to the case of Fig 34 & 36.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>76</td>
<td>27</td>
</tr>
</tbody>
</table>

After all the methods proposed above to handle the undesirable motion vectors, are incorporated into the proposed algorithm of Fig 33, the block diagram of the amended algorithm is as shown in Fig 39.
Figure 39. Block Diagram of BMA with Proposed Solutions to Handle Undesirable Motion Vectors Appearing in the Images from a UAV in Flight. The proposed solutions for various problems including uniformity in color, camera vibrations and shake, ego-motion and image noise, are incorporated with the block matching algorithm. The potential obstacles in the scene are thus identified by the presence of corrected motion vectors there. An obstacle avoidance maneuver is to be accordingly initiated.

The application of this algorithm on cases of Figs 34 and 35 are presented in Figs 38, 40 & 41. Besides these two cases, Figs. 42-49 present few more examples of application of this algorithm. Figs. 42-43 deal with detecting new obstacles entering the scene, which is identified by significant presence of motion vectors at the location of the new obstacle entering the scene (Fig. 42) and large numbers in the forward collision avoidance map (Fig.43). The second example is that of Figures 44 and 45, which present the case of identifying critical obstacles to be avoided, from the ones already present in
the scene, by having relative change in depth in scene, i.e. quickly approaching obstacles. Figs. 46-49 are similar examples with the camera mounted on a forward moving cart.

An overview of implementation platform is presented in Appendix D and the implemented code is attached as Appendix E.

Figure 40. Detecting Critical Obstacles in the Scene. The leader aircraft is changing altitude in the scene and is being identified by the significant presence of motion vectors there.

Figure 41. Forward Collision Avoidance Map for Detecting Critical Obstacles in the Scene. The leader aircraft is changing altitude in the scene and is being identified by the significant presence of motion vectors there. (For these specific results, the vectors in the 3rd row were suppressed being well below the horizon line and not on the collision course. Similarly the vectors generated from black image border being unrealistic were also suppressed). As before largest magnitudes give the maximum probability of the obstacle ahead on collision course (highlighted red) and minimum magnitudes guide us the obstacle free direction to turn to (highlighted green). Highlighted yellow are the numbers suggested to exercise caution in those directions.
Figure 42. Detecting New Obstacles Entering the Scene. A new obstacle (checkerboard) is entering the scene and is being identified by the significant presence of motion vectors there.

Figure 43. Forward Collision Avoidance Map for Detecting New Obstacles Entering the Scene. A new obstacle (checkerboard) is entering the scene and is being identified by the significant magnitude of motion vectors (value 163 highlighted red) in the matrix map. The minimum numbers (zeros – highlighted green) indicate the directions to turn to avoid collision ahead. Thus we need to turn slightly left and/or pitch up or down towards green zeros to avoid the obstacle highlighted in red.
Figure 44. **Identifying Critical Obstacles to be Avoided.** Relative depth to one of the objects (checkerboard) is decreasing rapidly in the scene. This indicates an obstacle quickly approaching nearby and is therefore critical. This object is identified by the significant presence of motion vectors there.

![Uncorrected motion vectors](image1.png) ![5th Image Corrected](image2.png) ![9th Image Corrected](image3.png) ![13th Image Corrected](image4.png) ![17th Image Corrected](image5.png) ![Total Corrected](image6.png)

Figure 45. **Forward Collision Avoidance Map for Identifying Critical Obstacles to be Avoided.** Relative depth to one of the objects (checkerboard) is decreasing rapidly in the scene. This indicates a nearby obstacle quickly approaching and is therefore critical. This object is identified by significant numbers in the matrix map (193 most significant in left top corner highlighted red, followed by 54 in the adjacent section). The minimum numbers (zeros – highlighted green) suggest pitch down motion and/or right turn to avoid collision ahead.

<table>
<thead>
<tr>
<th>193</th>
<th>54</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
Figure 46. Detecting Obstacles Entering or Moving in the Scene from a Moving Camera. An obstacle is entering and moving in the scene and is being identified by the significant presence of motion vectors there. Ceiling being closer to the moving obstacle gives larger motion vectors.

Figure 47. Forward Collision Avoidance Map for Detecting Obstacles Entering or Moving in the Scene from a Moving Camera. An obstacle is entering and moving in the scene and is being identified by the significant magnitude of motion vectors (values 6,7,9) in the matrix map. Ceiling being closer correctly generates larger motion (value 13). Minimum nubers (zeros – highlighted green) suggest, we need to turn left and pitch down to avoid obstacles ahead.
Figure 48. Identifying Critical Obstacles to be Avoided from a Moving Camera. With a camera mounted on a moving cart, the relative depth to one of the objects (checkerboard) is decreasing rapidly in the scene. This indicates an obstacle quickly approaching nearby and is therefore critical. This object is identified by the significant presence of motion vectors there.

<table>
<thead>
<tr>
<th>2</th>
<th>13</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 49. Forward Collision Avoidance Map for Identifying Critical Obstacles to be Avoided from a Moving Camera. With the camera mounted on a moving cart, the relative depth to one of the objects (checkerboard) is decreasing rapidly in the scene. This indicates a nearby obstacle quickly approaching and is therefore critical. This object is identified by significant numbers in the matrix map (22 most significant in forward left section, followed by 13 in the adjacent upper section).
Quantifying Forward Collision Avoidance Map

As mentioned above, the most significant number in the Forward Collision Avoidance Map indicates the most imminent danger ahead i.e. an obstacle quickly approaching, or an obstacle much closer to own-ship as compared to all other obstacles in the scene. Likewise the least significant number in this map indicates least possibility of obstacles ahead and suggests the directions to turn to, to avoid obstacles. Thus for example in Fig. 45 the number 193 indicates an obstacles that needs to be avoided in the left top corner of the scene ahead. The zeros indicate, we need to duck down slightly and/or to turn right slightly to avoid that collision.

However, the numbers in this map may be utilized further to extract more detailed information. Thus for example, we could find minima, maxima, mean and standard deviation values for these 12 numbers. Then the most significant number if is close to the mean, is not as significant a danger, as compared to the the most significant number that is beyond 2 or 3 times the standard deviation higher than the mean.

Likewise, the least significant number in the map, is if too close to the mean, may not mean a completely obstacle free path ahead and caution needs to be exercised turning towards that direction in order to avoid obstacles. If however, this number is lower than 2 or 3 times the standard deviation from the mean, this would mean a relative obstacle free path ahead.

Comparisons could be made from absolute zero value, instead of relative information generated in these maps (relative information was generated primarily to correct for own-ship motion). In such a case, various diversifeid scenarios could be tested to quantify the absolute numbers and corresponding estimate of depth of objects in the scene. With this estimate of depth, even the most significant number found in the map, may not necessitate an immediate collision avoidance maneuver, if sufficient depth exists to allow uninterrupted forward motion ahead for a while.
CHAPTER 14
CONCLUSION

Object detection using images from a single camera for search and identification problem was attempted first. A vision-based algorithm was presented to detect a visual marker or a beacon in real time and in the presence of excessive colored noise, as well as with interference from the onboard avionics. Various image analysis techniques including color histograms, filtering techniques and color space analyses were used to establish typical pixel-based characteristics of the desired visual marker or a beacon in an image. A block based search algorithm was used to search for those established characteristics in a real-time image data stream from a colored camera. A very low cost noise and interference filter was implemented in order to handle excessive noise that was encountered during flight tests. The final algorithm used both multiple thresholding and linear confidence level calculation in a full search block based approach and was demonstrated to handle as high a noise and interference as 30-40% in the image data. This work was implemented on the GTAR lama aircraft and was successfully used in International Aerial Robotics competition 2009 by Association for Unmanned Vehicle Systems International, to detect a blue light emitting diode in real time.

Next, the problem of object detection using images from a single camera for collision avoidance was attempted. Full 3D reconstruction approach was used and it was demonstrated that it can successfully generate a 3D model of the scene from 2D image information. This information was sufficient to initiate a collision avoidance maneuver. However, for its practical implementation on various UAVs/robots, an in-depth analysis revealed various constraints associated with this technique.

Such detailed analysis of 3D construction approach and various other approaches to solve the collision avoidance problem using images from a single camera, led to
proposing a new technique of motion estimation using a block matching algorithm, (which is otherwise common to video compression codecs). This technique was successfully applied to detect potential obstacles in the scene, first on synthetically generated images, and then on images from a UAV on ground. The proposed technique was then implemented on videos from a UAV in actual flight, in which various undesirable motion vectors were encountered. The work further presented an analysis of these undesirable motion vectors and proposed and implemented various methodologies to handle these undesirable vectors. Thus potential obstacles, whether already in the scene, or newly entering, were demonstrated to be successfully detected, leading to initiation of a collision avoidance maneuver.

Contributions of the work presented in this dissertation, in terms of accepted or published research are enlisted in Appendix F [76-81].

This research may be regarded as the first attempt to use motion estimation for collision avoidance. As a future work it is recommended that the proposed approach be thoroughly tested, improved and subsequently utilized for various UAV platforms. If need be, faster algorithms instead of Exhaustive Full Search may be used with BMA. The examples of such algorithms were quoted in Chapter 12 [23-26].
APPENDIX A

DERIVATIONS FOR 3D RECONSTRUCTION EQUATIONS

Derivation of Eq. (19)

Referring Eqs. (7) and (11), the residual vector on 2D image plane is given by

\[
d = \begin{bmatrix} dx \\ dy \end{bmatrix} = Z - x_k = \begin{bmatrix} z_i \\ z_j \\ z_k \end{bmatrix} - \begin{bmatrix} y_k \\ z_k \end{bmatrix}
\]  

(A.1)

where \(Z\) is observed feature point on image plane and \(x_k\) is the projected feature point from 3D space on 2D image plane. Hence,

\[
\frac{\partial d}{\partial x_k} = \begin{bmatrix} \frac{\partial dx}{\partial y_k} & \frac{\partial dx}{\partial z_k} \\ \frac{\partial dy}{\partial y_k} & \frac{\partial dy}{\partial z_k} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = -I_2
\]

This is one of the results as required for Eq. (18) and used in Eq. (19). The second result is derived as follows. Referring Eq. (8), the relative position vector of the camera is

\[
X = X_v - X_p
\]

\[
X = \begin{bmatrix} X_{ck} \\ Y_{ck} \\ Z_{ck} \end{bmatrix} = \begin{bmatrix} X_v \\ Y_v \\ Z_v \end{bmatrix} - \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix}
\]

Referring Eq. (11), the corner feature projected onto 2D image plane from 3D space is given by
\[ x_k = \begin{bmatrix} y_k \\ z_k \end{bmatrix} = \frac{f}{X_{ck}} \begin{bmatrix} Y_{ck} \\ Z_{ck} \end{bmatrix} = \begin{bmatrix} Y_{ck} \\ X_{ck} \\ Z_{ck} \\ X_{ck} \end{bmatrix} \]

where unit focal length is supposed without loss of generality, \( y_k, z_k \) indicate 2D coordinates in image plane and \( X_{ck}, Y_{ck}, Z_{ck} \) indicate coordinates in 3D world. Hence,

\[
\frac{\partial x_k}{\partial X} = \begin{bmatrix} \frac{\partial y_k}{\partial X_v} & \frac{\partial y_k}{\partial Y_v} & \frac{\partial y_k}{\partial Z_v} \\ \frac{\partial z_k}{\partial X_v} & \frac{\partial z_k}{\partial Y_v} & \frac{\partial z_k}{\partial Z_v} \end{bmatrix}
\]

\[
= \begin{bmatrix} \frac{\partial}{\partial X_v} \left( \frac{Y_v - Y_p}{X_v - X_p} \right) & \frac{\partial}{\partial Y_v} \left( \frac{Y_v - Y_p}{X_v - X_p} \right) & \frac{\partial}{\partial Z_v} \left( \frac{Y_v - Y_p}{X_v - X_p} \right) \\ \frac{\partial}{\partial X_v} \left( \frac{Z_v - Z_p}{X_v - X_p} \right) & \frac{\partial}{\partial Y_v} \left( \frac{Z_v - Z_p}{X_v - X_p} \right) & \frac{\partial}{\partial Z_v} \left( \frac{Z_v - Z_p}{X_v - X_p} \right) \end{bmatrix}
\]

\[
= \begin{bmatrix} \frac{-(Y_v - Y_p)}{(X_v - X_p)^2} & \frac{(X_v - X_p)}{(X_v - X_p)^2} & 0 \\ \frac{-(Z_v - Z_p)}{(X_v - X_p)^2} & 0 & \frac{-(X_v - X_p)}{(X_v - X_p)^2} \end{bmatrix}
\]

\[
= \frac{1}{X_{ck}} \begin{bmatrix} -Y_{ck} \\ X_{ck} \\ -Z_{ck} \\ X_{ck} \end{bmatrix} = \frac{1}{X_{ck}} \begin{bmatrix} -x_k \\ I_2 \end{bmatrix}
\]

which is the required result as used in Eq. (19).
Derivation for Eqs. (21) and (22)

Referring Eq (A.1) given above,

\[
d = \begin{bmatrix} dx \\ dy \end{bmatrix} = Z - x_k = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} - \begin{bmatrix} y_k \\ z_k \end{bmatrix}
\]

Hence,

\[
\frac{\partial d}{\partial Z} = \begin{bmatrix} \frac{\partial dx}{\partial z_1} & \frac{\partial dx}{\partial z_2} \\ \frac{\partial dy}{\partial z_1} & \frac{\partial dy}{\partial z_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2
\]

which is the required result.

Derivation for Eqs. (23) and (24)

Referring Eqs. (5) and Eq. (A.1),

\[J = dx^2 + dy^2\]

\[
d = \begin{bmatrix} dx \\ dy \end{bmatrix}
\]

\[
\frac{\partial J}{\partial d} = \begin{bmatrix} \frac{\partial J}{\partial dx} & \frac{\partial J}{\partial dy} \end{bmatrix} = \begin{bmatrix} 2dx \\ 2dy \end{bmatrix} = 2 \begin{bmatrix} dx \\ dy \end{bmatrix}^T = 2d^T
\]

which is the required result as used in Eqs. (23) and (24).
APPENDIX B

OVERVIEW OF CHosen CAMERA (SONY FCB-EX980S) WITH IMAGE STABILIZATION [74]

FCB-EX Series

Sony’s new FCB-EX series of color block cameras represent an evolution in security dome, police vehicle and traffic monitoring applications. These high-sensitivity FCB-EX color block cameras are equipped with an integrated lens and incorporate such new and unique surveillance features such as: Enhanced Spherical Privacy Zone Masking Function for use with a pan/tilt system and Sony’s SMART Lens Control Technology. In addition, these FCB-EX cameras also incorporate high-performance Digital Signal Processing (DSP) that provides for enhanced picture quality and operability compared to conventional block cameras.

The FCB-EX9805/EX980P color block cameras are equipped with an incredibly high 25x zoom lens with a wide horizontal field of view, ideal for use in security domes and traffic monitoring applications. The FCB-EX9805/EX980P cameras combine a 1/3” Super HAD™ CCD with Sony’s new powerful 25x zoom “SMART” lens with a high telephoto zoom capability and image stabilizer function, allowing users to zoom in on small or distant objects with exceptional clarity and stability. The FCB-EX9805/EX980P cameras combine a high-sensitivity 1/3” Exview HAD™ CCD and Sony’s new powerful 25x zoom “SMART” lens with a wide horizontal field of view.

The FCB-EX480C/EX480P color block cameras are equipped with an 18x zoom lens and incorporate a high-sensitivity 1/3” Exview HAD™ CCD allowing images to be captured a minimum illumination of 0.7 lux.

In addition, all of these FCB-EX cameras are equipped with a variety of convenient functions such as E-Flip, Alarm, Picture Freeze and Auto ICR which have been inherited from earlier FCB-EX Series cameras. Moreover, all of these FCB-EX cameras use lead-free solder and halogen-free printed circuit boards, making these cameras environmentally friendly.

Combining superb picture quality and a variety of unique and convenient features, these new FCB-EX cameras are the perfect match, both indoor or outdoor, for demanding security and monitoring applications.

- 25x Optical Zoom Capability - FCB-EX9805/EX980P and FCB-EX980/EX980P
- 18x Optical Zoom Capability - FCB-EX480C/EX480P and FCB-EX480C/EX480P
- Enhanced Spherical Privacy Zone Masking Function (max. 24 masking blocks)
- SMART Lens Control (Sony Modular Automatic Lens Reset Technology)
- Electronic-Flip (E-Flip) Function
- Alarm Function
- Auto ICR (IR Cut- filter Removal) Mode
- Picture Freeze Function
- Image Stabilizer
- High-Performance Digital Signal Processing (DSP)
- High-Speed Serial Interface (max. 38.4 Kbps) with TIL Signal-Level Control (VSCA™ protocol)
- Equipped with Key Switch Connector (CN601) for Camera Control with External Equipment
- Various Customizable Settings
- Internal/External Sync
- Low Power Consumption (1.6 W with motors inactive)
- EEPROM Backup System without Battery
- 16-bytes of Free Memory is Available for Recording Data such as Product Serial Numbers and Camera System ID Numbers
- Lead-free Solder and Halogen-free Printed-circuit-boards

*1 FCB-EX805/EX805P, FCB-EX980/EX980P, FCB-EX480C/EX480P
*2 FCB-EX805/EX805P, FCB-EX980/EX980P, FCB-EX480C/EX480P
*3 FCB-EX805/EX805P only
FCB-EX SERIES LINE-UP

FCB-EX980S/EX980SP
FCB-EX980/EX980P

- 1/4-type Super HAD CCD (FCB-EX980S/EX980SP)
- 1/4-type ExView HAD CCD (FCB-EX980/EX980P)
- 312x Zoom Ratio (25x optical, 12x digital)
- Enhanced Spherical Privacy Zone Masking Function
- Image Stabilizer Function (FCB-EX980S/EX980SP)
- Minimum Illumination of 1.0 lx (typical) (FCB-EX980/EX980P)
- E-Flip Function
- Alarm Function
- Spot AE
- Auto ICR (IR Cut-filter Removal) Mode
- Picture Freeze Function
- Key Switch Connector (CN601) for Camera Control with External Equipment
- Electronic Shutter/Slow Shutter
- High-Speed Serial Interface (maximum 38.4 kbps) with TTL Signal-Level Control (VISCA protocol)
- Internal/External Sync

The Image Stabilizer Function

When the Image Stabilizer Function is ON, it helps in obtaining a stable image free of vibration caused by jarring movements. For a vibration frequency of around 10 Hz, correction is approximately 90%.

Unit: mm
# SPECIFICATIONS

<table>
<thead>
<tr>
<th>Feature</th>
<th>FCB-SX9005</th>
<th>FCB-SX9005P</th>
<th>FCB-SX810</th>
<th>FCB-SX810P</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image device</strong></td>
<td>1/4-type Super HAD CCD</td>
<td>1/4-type Super HAD CCD</td>
<td>1/4-type Exmor HAD CCD</td>
<td>1/4-type Exmor HAD CCD</td>
</tr>
<tr>
<td><strong>Signal System</strong></td>
<td>NTSC</td>
<td>PAL</td>
<td>NTSC</td>
<td>PAL</td>
</tr>
<tr>
<td><strong>Effective picture elements</strong></td>
<td>Approx. 680,000 pixels</td>
<td>Approx. 680,000 pixels</td>
<td>Approx. 380,000 pixels</td>
<td>Approx. 380,000 pixels</td>
</tr>
<tr>
<td><strong>Lens</strong></td>
<td>f=5.5 mm (wide end) to 91.0 mm (tele end)</td>
<td>f=5.5 mm (wide end) to 91.0 mm (tele end)</td>
<td>f=5.5 mm (wide end) to 91.0 mm (tele end)</td>
<td>f=5.5 mm (wide end) to 91.0 mm (tele end)</td>
</tr>
<tr>
<td><strong>Digital zoom</strong></td>
<td>5x (12x with optical zoom)</td>
<td>12x (312x with optical zoom)</td>
<td>12x (312x with optical zoom)</td>
<td>12x (312x with optical zoom)</td>
</tr>
<tr>
<td><strong>Viewing angle (H)</strong></td>
<td>42.2° (wide end) to 1.5° (tele end)</td>
<td>42.2° (wide end) to 1.5° (tele end)</td>
<td>55° (wide end) to 2,27° (tele end)</td>
<td>55° (wide end) to 2,27° (tele end)</td>
</tr>
<tr>
<td><strong>Minimum working distance</strong></td>
<td>320 mm (wide end) to 1530 mm (tele end)</td>
<td>320 mm (wide end) to 1530 mm (tele end)</td>
<td>320 mm (wide end) to 1530 mm (tele end)</td>
<td>320 mm (wide end) to 1530 mm (tele end)</td>
</tr>
<tr>
<td><strong>Sync system</strong></td>
<td>Internal/External (V-Lock)</td>
<td>Internal/External (V-Lock)</td>
<td>Internal/External (V-Lock)</td>
<td>Internal/External (V-Lock)</td>
</tr>
<tr>
<td><strong>Minimum Illumination</strong></td>
<td>2.0 lx (typical) (F1.6, 50 IRE)</td>
<td>2.0 lx (typical) (F1.6, 50 IRE)</td>
<td>1.0 lx (typical) (F1.6, 50 IRE)</td>
<td>1.0 lx (typical) (F1.6, 50 IRE)</td>
</tr>
<tr>
<td><strong>SN ratio</strong></td>
<td>More than 50 dB</td>
<td>More than 50 dB</td>
<td>More than 50 dB</td>
<td>More than 50 dB</td>
</tr>
<tr>
<td><strong>Electronic shutter</strong></td>
<td>1/60 to 1/2,000 sec, 22 steps</td>
<td>1/60 to 1/2,000 sec, 22 steps</td>
<td>1/60 to 1/2,000 sec, 22 steps</td>
<td>1/60 to 1/2,000 sec, 22 steps</td>
</tr>
<tr>
<td><strong>FV compensation</strong></td>
<td>10.5 to + 16.5 dB (1.5 dB steps)</td>
<td>10.5 to + 16.5 dB (1.5 dB steps)</td>
<td>10.5 to + 16.5 dB (1.5 dB steps)</td>
<td>10.5 to + 16.5 dB (1.5 dB steps)</td>
</tr>
<tr>
<td><strong>Privacy zone masking</strong></td>
<td>On/Off (4 positions)</td>
<td>On/Off (4 positions)</td>
<td>On/Off (4 positions)</td>
<td>On/Off (4 positions)</td>
</tr>
<tr>
<td><strong>Finder cancel</strong></td>
<td>Auto</td>
<td>Auto</td>
<td>Auto</td>
<td>Auto</td>
</tr>
<tr>
<td><strong>Picture effects</strong></td>
<td>E-Flip, NeoArt, BlackWhite, Mirror Image</td>
<td>E-Flip, NeoArt, BlackWhite, Mirror Image</td>
<td>E-Flip, NeoArt, BlackWhite, Mirror Image</td>
<td>E-Flip, NeoArt, BlackWhite, Mirror Image</td>
</tr>
<tr>
<td><strong>Camera operation switch</strong></td>
<td>Zoom, tele, zoom wide</td>
<td>Zoom, tele, zoom wide</td>
<td>Zoom, tele, zoom wide</td>
<td>Zoom, tele, zoom wide</td>
</tr>
<tr>
<td><strong>Video output</strong></td>
<td>Y/C: 1.0Vp-p (sync negative), Y/C Output</td>
<td>Y/C: 1.0Vp-p (sync negative), Y/C Output</td>
<td>Y/C: 1.0Vp-p (sync negative), Y/C Output</td>
<td>Y/C: 1.0Vp-p (sync negative), Y/C Output</td>
</tr>
<tr>
<td><strong>Camera control interface</strong></td>
<td>V/SCA (TTL, signal level), baud rate: 19.2 kbps, 9.6 kbps, 38.4 kbps, 1 or 2 step bit selectable</td>
<td>V/SCA (TTL, signal level), baud rate: 19.2 kbps, 9.6 kbps, 38.4 kbps, 1 or 2 step bit selectable</td>
<td>V/SCA (TTL, signal level), baud rate: 19.2 kbps, 9.6 kbps, 38.4 kbps, 1 or 2 step bit selectable</td>
<td>V/SCA (TTL, signal level), baud rate: 19.2 kbps, 9.6 kbps, 38.4 kbps, 1 or 2 step bit selectable</td>
</tr>
<tr>
<td><strong>Storage temperature</strong></td>
<td>-4 to +140°F (-20 to 60°C)</td>
<td>-4 to +140°F (-20 to 60°C)</td>
<td>-4 to +140°F (-20 to 60°C)</td>
<td>-4 to +140°F (-20 to 60°C)</td>
</tr>
<tr>
<td><strong>Operating temperature</strong></td>
<td>22 to 122°F (0 to +50°C)</td>
<td>22 to 122°F (0 to +50°C)</td>
<td>22 to 122°F (0 to +50°C)</td>
<td>22 to 122°F (0 to +50°C)</td>
</tr>
<tr>
<td><strong>Power consumption</strong></td>
<td>6 to 12 V DC, 1.6 W motors (inactive), 2.3 W (motors active)</td>
<td>6 to 12 V DC, 1.6 W motors (inactive), 2.3 W (motors active)</td>
<td>6 to 12 V DC, 1.6 W motors (inactive), 2.3 W (motors active)</td>
<td>6 to 12 V DC, 1.6 W motors (inactive), 2.3 W (motors active)</td>
</tr>
<tr>
<td><strong>Weight</strong></td>
<td>8.1 oz (230 g)</td>
<td>8.1 oz (230 g)</td>
<td>8.1 oz (230 g)</td>
<td>8.1 oz (230 g)</td>
</tr>
<tr>
<td><strong>Dimensions (WxHxD)</strong></td>
<td>2 1/4 x 2 1/4 x 3 1/4 inches (52.0 x 57.5 x 88.5 mm)</td>
<td>2 1/4 x 2 1/4 x 3 1/4 inches (52.0 x 57.5 x 88.5 mm)</td>
<td>2 1/4 x 2 1/4 x 3 1/4 inches (52.0 x 57.5 x 88.5 mm)</td>
<td>2 1/4 x 2 1/4 x 3 1/4 inches (52.0 x 57.5 x 88.5 mm)</td>
</tr>
</tbody>
</table>

©2005 Sony Corporation. All rights reserved. Reproduction in whole or in part without written permission is prohibited. Design, features and specifications are subject to change without notice. All non-metric weights and measures are approximate. Sony, SonyView HAD, Super HAD and VISCA are trademarks of Sony.

Sony Electronics Inc.
1 Sony Drive
Park Ridge, NJ 07656
201-930-7000
www.sony.com/video_cameras
IS-1191
MX1019SV1

Printed in USA (2005)
APPENDIX C

SELECTING APPROPRIATE VIBRATION INSULATORS [75]

Selection procedure for appropriate vibration insulators is described below:

<table>
<thead>
<tr>
<th>Selection step</th>
<th>Applied to our case</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Determine the load that each mount will bear when supporting the equipment</td>
<td>Chosen Sony FCB-980S camera weighs 0.5lbs. Having 4 mounts means, each mount takes 0.13lbs.</td>
</tr>
<tr>
<td>weight. Total weight divided by the number of mounting positions is the load for each mount. This is only true when having even weight distribution. Otherwise, distribute weight accordingly.</td>
<td></td>
</tr>
<tr>
<td>2. Determine the lowest forcing frequency of the vibration source to be supported by the mounts. This is usually equal to the operating speed in revolutions per minute.</td>
<td>The engine onboard Yak-54 is DA100 and it operates at 2000-7600 rpm. Normal operating rpm is around 4500.</td>
</tr>
<tr>
<td>3. Choose the percent isolation that will be satisfactory for the purpose. Except for special cases, 81% isolation is generally considered satisfactory.</td>
<td>81% vibration isolation chosen.</td>
</tr>
<tr>
<td>4. Referring to the Basic Vibration Chart below (Fig 50), find the static deflection for the forcing frequency (Step 2, above) at the chosen percent isolation (Step 3). Note that a mount must give at least this minimum static deflection, with the specific load applied, to provide the desired isolation.</td>
<td>Low frequencies are critical, as these necessitate more deflection required from the mounts. For 2000rpm, a static deflection required from the mount for 81% vibration isolation is 0.06in.</td>
</tr>
<tr>
<td>5. Select the mount series with the physical features (shape, attachment facilities, “fail-safe&quot; safety feature, load range, etc.) required by the application.</td>
<td>Silicone Gel Type mounts (V10Z61M series) Fig 51 and Urethane cylindrical mounts (V10Z60 series) Fig 52 are short-listed.</td>
</tr>
<tr>
<td>6. Then proceed as follows:</td>
<td></td>
</tr>
<tr>
<td>a) Having selected the mount series, refer to the individual styles, and note the styles whose maximum loads are greater than the load each mount is to carry.</td>
<td></td>
</tr>
<tr>
<td>b) Referring to the load deflection graphs of the</td>
<td></td>
</tr>
</tbody>
</table>
styles likely to be chosen, locate the applied load value (Step 1, above) on the appropriate graph; i.e., compression and/or shear.

c) Moving horizontally to the right on the graph, locate the point of intersection with the minimum static deflection found in step 4.

d) Mounts with curves above this point of intersection cannot be used, as the load (Step 1) is not sufficient to produce the required minimum deflection (Step 4).

e) Mounts with curves below the point of intersection can be used, as the given load will produce deflection greater than the minimum required. Note, however, that if the applied load is above the line x--x on a curve, the mount is not recommended for this static load.

More than one style may have load-deflection curves that are suitable. The final selection can depend on other requirements such as the cost of the mounts, possible increased load requirements in the future, relative advantage of additional isolation, space available for the mounts, constraints on allowable deflection, attachment requirements, etc. However, in the absence of any overriding consideration, usually the mount selected has its curve closest to the point of intersection (Step 6c); i.e., the mount with the minimum deflection at the applied load.

7. Select the mount that is designed to operate in your temperature range and environment.

For our case if deflection
\[ x = \frac{F}{k} = \frac{0.13}{25} = 0.005\text{in} \]
is good only above 5000rpm

Silicone Gel Type mounts have the right weight limit, static deflection and vibration isolation. However, they could give resonance between 2520-3840rpm [75].

Urethane cylindrical mounts have no resonance problems. However, these are most suited for heavier loads than the Sony FCB-EX camera. Hence will not give enough vibration deflection for vibration isolation at low rpms.

Both urethane cylindrical mounts and gel type mounts are not suitable for low rpms (<4500). Urethane mounts give insufficient vibration isolation and gel type mounts give resonance at such low rpms

Urethane cylindrical mounts are chosen as having insufficient vibration isolation is far better than having resonance at low rpms.
Fig 50. Vibration Frequency vs. Static Deflection vs Isolation Efficiency Values[75]. This plot was used to choose most suitable vibration isolators for implementation of the work proposed in this dissertation.
Fig 51. Silicone Gel Type Mounts [75]. These mounts were identified as one of the prospective mounts for vibration isolation of the camera installed on the aircraft.
Fig 52. Urethane Cylindrical Mounts [75]. These mounts were found better than silicone gel type mounts and were finally used.
APPENDIX D

OVERVIEW OF MOTION ESTIMATION IMPLEMENTATION

PLATFORM

Overview of Chosen UAV Platform : GT-YAK

The chosen aircraft for implementation is GT Yak. It is a scaled down model of actual Yakovlev Yak-54 which is 1990s Russian aerobatic and sport competition aircraft [72], Fig 53. The GT Yak has been developed from Quique Somenzini Yak 54 – 102 and is shown in Fig 54.

Fig 53. Yak-54 Aircraft. Front, left side and top views shown.
Fig 54. GT Yak UAV. GT-Yak is a scaled down model of actual Yak-54 aircraft. It has been developed at Georgia Tech from Quique Somenzini Yak 54 - 102. Shown above is GT-Yak in a flight test.

Significant specifications of GT-Yak are as follows:

- **Engine** = Model DA-100, 100cc, (10 hp), RPM 2000-7600
- **Aircraft Length** = 95 in
- **Wing Span** = 102 in
- **Wing Area** = 1975 sq in
- **Weight** = 26.5 lbs (stock configuration)
- **Standard Controls** = Throttle, Elevator, Rudder, Ailerons

**Overview of Implementation and Integration Software/Hardware**

GT-YAK carries a Flight Control System (FCS20) and an onboard computer for vision or other tasks. A brief description of FCS20 referring [73], is as follows.

The Flight Control System 20 (FCS20) is a compact, self-contained Guidance, Navigation, and Control system that has been developed to enable advanced autonomous behavior in a wide range of Unmanned Aerial Vehicles (UAVs). The FCS20 uses a floating point Digital Signal Processor (DSP) for high level serial processing, a Field
Programmable Gate Array (FPGA) for low level parallel processing, and GPS and Micro Electro Mechanical Systems (MEMS) sensors. In addition to guidance, navigation, and control functions, the FCS20 is capable of supporting advanced algorithms such as automated reasoning, artificial vision, and multi-vehicle interaction. It can also support payload control, image processing, communications interfaces (encryption), vehicle health monitoring, and other high level algorithms. Using a single FCS20 to support these systems differs from the more traditional approach of using separate and dedicated hardware components. The data flow diagram is presented in Fig.55. Further details of its computing, communications and information aspects are given in [73].

**Fig.55.** FCS20 Data Flow Diagram [73]. It is a self-contained guidance, navigation & control system for autonomous UAVs.
APPENDIX E

IMPLEMENTED C++ CODE FOR

PROPOSED MOTION ESTIMATION

/*

OBSTACLE DETECTION VIA MOTION VECTORS

Code uses Block Matching Algorithm with Exhaustive Search.

Requires 17 input images.
The code is stand-alone even for own ship motion corrections and
except for input images does not need any information from onboard systems.

It handles noise and unwanted motion vectors by using following techniques:
  Texture filter (for uniform color correction)
  Ego-motion correction (for own-ship motion correction)
  Temporal accumulation (to handle noise)
  Spatial accumulation (to handle noise)

Results are given in three 3x4 obstacle avoidance or guidance maps for obstacles ahead,
with highest numbers indicating max probability of obstacles ahead,
and lowest numbers or zeros, guiding us where to turn.
(Three separate maps are displayed
  one after forward motion correction,
  second after separate vertical-lateral-roll motion correction,
  and third after combined corrections.)

Change 5/5/10: #define varth=17, fwdweightage line 1307, normalization by min motion line 419

Written: Spring 2010 by Syed Irtiza Ali Shah
Advisor: Dr Eric N Johnson

OpenCV reference manual:
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <cxcore.h>
#include <cvaux.h>
#include <conio.h>

// Input images

#define IMG_FILENAME1 "001.png"
#define IMG_FILENAME2 "002.png"
#define IMG_FILENAME3 "003.png"
#define IMG_FILENAME4 "004.png"
#define IMG_FILENAME5 "005.png"
#define IMG_FILENAME6 "006.png"
#define IMG_FILENAME7 "007.png"
#define IMG_FILENAME8 "008.png"
#define IMG_FILENAME9 "009.png"
#define IMG_FILENAME10 "010.png"
#define IMG_FILENAME11 "011.png"
#define IMG_FILENAME12 "012.png"
#define IMG_FILENAME13 "013.png"
#define IMG_FILENAME14 "014.png"
#define IMG_FILENAME15 "015.png"
#define IMG_FILENAME16 "016.png"
#define IMG_FILENAME17 "017.png"
//Global Parameters for code
//=============================================================================
#define MBSIZE 8 //square Mega block size
#define PPP 12 // search depth parameter (search square window is MBSIZE+2P)
#define VARTH 17 //threshold variance for texture filter (10-50 is good)
#define DELTA 0.05 //5% image noise for fwd motion corrected avoidance map
#define IMG_ROWS 240
#define IMG_COLS 320
#define MAXCOST 65537 //arbitrary maximum supposed cost value
//Note: For image size 480x640, use MBSIZE as 16 and PPP as 18

//Declare functions, structures & global variables
//=============================================================================

//Error Handler
int MyErrorHandler(int status, const char* func_name, const char* err_msg, const char* file_name, int line, void*)
{
    printf( "%s\n", err_msg);
    _getch();
    return 0;
}

int num_vectors=IMG_ROWS*IMG_COLS/(MBSIZE*MBSIZE); //number of vectors
int mv_rows=5;

//Image cropping function
IplImage* Crop(IplImage*, int, int, int, int); // crops image to a desired size image block
//Motion estimation function
void motionEstES2(float** vectors, IplImage*, IplImage*); //calculates motion vectors
//Mean absolute difference calculator
float costFuncMAD(float**,float**, int); //calculates cost: mean abs diff b/w blocks

//Minimum cost and its vector structure
struct minCostfound //structure to output min cost found and its location vector
{
    // in the cost matrix
    int dx, dy;
    float min;
};

//Minimum Cost finding function
minCostfound minCost(float**); //function to search/return minCostfound

//Forward and vert-lat motion correction structure
struct detectedObstacles //structure to output map (fwd motion corrected) and
{
    float Norm[3][4]; // norm (lat-vert motion corrected matrix)
    float ExpObsMap[3][4];
};

//Function to detect obstacles after own-ship motion correction
detectedObstacles detectObstacles(float**, int, int); //function to return map&norm


//************************************************************************************
//         MAIN FUNCTION
//************************************************************************************

void main(int argc, char *argv[])
{
    cvSetErrMode(CV_ErrModeParent);
    cvRedirectError(MyErrorHandler);
    int firstdim =2;
    int i, j;
    //Allocate memory for motion vectors between images
    float** vectors = new float*[firstdim]; //to return from motion estimation function
float** motionVect21 = new float*[firstdim]; //vectors between image2 and image1
float** motionVect32 = new float*[firstdim];
float** motionVect43 = new float*[firstdim];
float** motionVect54 = new float*[firstdim];
float** motionVect65 = new float*[firstdim];
float** motionVect76 = new float*[firstdim];
float** motionVect87 = new float*[firstdim];
float** motionVect98 = new float*[firstdim];
float** motionVect109 = new float*[firstdim];
float** motionVect1110 = new float*[firstdim];
float** motionVect1211 = new float*[firstdim];
float** motionVect1312 = new float*[firstdim];
float** motionVect1413 = new float*[firstdim];
float** motionVect1514 = new float*[firstdim];
float** motionVect1615 = new float*[firstdim];
float** motionVect1716 = new float*[firstdim];
float** motionVect1701 = new float*[firstdim]; //direct between img17 and img1
float** motionVectA  = new float*[firstdim]; //For temporal accumulation of vectors
float** motionVectB  = new float*[firstdim];
float** motionVectC  = new float*[firstdim];
float** motionVectD  = new float*[firstdim];
float** mvTotal   = new float*[firstdim];

for (i=0;i<firstdim;i++)
{
    vectors[i]   = new float[num_vectors];
    motionVect21[i]  = new float[num_vectors]; //between img2 and img1
    motionVect32[i]  = new float[num_vectors]; //between img3 and img2
    motionVect43[i]  = new float[num_vectors];
    motionVect54[i]  = new float[num_vectors];
    motionVect65[i]  = new float[num_vectors];
    motionVect76[i]  = new float[num_vectors];
    motionVect87[i]  = new float[num_vectors];
    motionVect98[i]  = new float[num_vectors];
    motionVect109[i] = new float[num_vectors];
    motionVect1110[i] = new float[num_vectors];
    motionVect1211[i] = new float[num_vectors];
    motionVect1312[i] = new float[num_vectors];
    motionVect1413[i] = new float[num_vectors];
    motionVect1514[i] = new float[num_vectors];
    motionVect1615[i] = new float[num_vectors];
    motionVect1716[i] = new float[num_vectors];
    motionVect1701[i] = new float[num_vectors]; //between img17 & img1
    motionVectA[i]  = new float[num_vectors];
    motionVectB[i]  = new float[num_vectors];
    motionVectC[i]  = new float[num_vectors];
    motionVectD[i]  = new float[num_vectors];
    mvTotal[i]   = new float[num_vectors];
}

//Create Image variables
IplImage *img1=0, *img2=0, *img3=0, *img4=0, *img5=0, *img6=0;
IplImage *img7=0, *img8=0, *img9=0, *img10=0, *img11=0, *img12=0;
IplImage *img13=0, *img14=0, *img15=0, *img16=0, *img17=0;

int mv_rows, mv_cols;

mv_rows = int(IMG_ROWS/MBSIZE); //for block motion vectors
mv_cols = int(IMG_COLS/MBSIZE);

//allocate memory for components and magnitude of motion vectors
float** v     = new float*[mv_rows]; //row allocations
float** u     = new float*[mv_rows];
float** magnitude = new float*[mv_rows];
float** vA    = new float*[mv_rows];
float** uA    = new float*[mv_rows];
float** magnitudeA = new float*[mv_rows];
float** vB    = new float*[mv_rows];
float** uB    = new float*[mv_rows];
float** magnitudeB = new float*[mv_rows];
float** vC    = new float*[mv_rows];
float** uC    = new float*[mv_rows];
float** magnitudeC = new float*[mv_rows];
float** vD    = new float*[mv_rows];
float** uD    = new float*[mv_rows];
float** magnitudeD = new float*[mv_rows];
float** vTotal = new float*[mv_rows];
float** uTotal = new float*[mv_rows];
float** magTotal  = new float*[mv_rows];
for (i=0;i<mv_rows;i++)
{
    v[i]     = new float[mv_cols];
    u[i]     = new float[mv_cols];
    magnitude[i]   = new float[mv_cols];
    vA[i]     = new float[mv_cols];
    uA[i]    = new float[mv_cols];
    magnitudeA[i]   = new float[mv_cols];
    vB[i]     = new float[mv_cols];
    uB[i]     = new float[mv_cols];
    magnitudeB[i]   = new float[mv_cols];
    vC[i]     = new float[mv_cols];
    uC[i]     = new float[mv_cols];
    magnitudeC[i]   = new float[mv_cols];
    vD[i]     = new float[mv_cols];
   uD[i]     = new float[mv_cols];
    magnitudeD[i]   = new float[mv_cols];
    vTotal[i]   = new float[mv_cols];
    uTotal[i]   = new float[mv_cols];
    magTotal[i]   = new float[mv_cols];
}

//Load images (flag 0 ensures one grayscale channel only)
//===============================================================================
img1 = Crop(cvLoadImage(IMG_FILENAME1,0),0,0,IMG_ROWS,IMG_COLS);
img2 = Crop(cvLoadImage(IMG_FILENAME2,0),0,0,IMG_ROWS,IMG_COLS);
img3 = Crop(cvLoadImage(IMG_FILENAME3,0),0,0,IMG_ROWS,IMG_COLS);
img4 = Crop(cvLoadImage(IMG_FILENAME4,0),0,0,IMG_ROWS,IMG_COLS);
img5 = Crop(cvLoadImage(IMG_FILENAME5,0),0,0,IMG_ROWS,IMG_COLS);
img6 = Crop(cvLoadImage(IMG_FILENAME6,0),0,0,IMG_ROWS,IMG_COLS);
img7 = Crop(cvLoadImage(IMG_FILENAME7,0),0,0,IMG_ROWS,IMG_COLS);
img8 = Crop(cvLoadImage(IMG_FILENAME8,0),0,0,IMG_ROWS,IMG_COLS);
img9 = Crop(cvLoadImage(IMG_FILENAME9,0),0,0,IMG_ROWS,IMG_COLS);
img10 = Crop(cvLoadImage(IMG_FILENAME10,0),0,0,IMG_ROWS,IMG_COLS);
img11 = Crop(cvLoadImage(IMG_FILENAME11,0),0,0,IMG_ROWS,IMG_COLS);
img12 = Crop(cvLoadImage(IMG_FILENAME12,0),0,0,IMG_ROWS,IMG_COLS);
img13 = Crop(cvLoadImage(IMG_FILENAME13,0),0,0,IMG_ROWS,IMG_COLS);
img14 = Crop(cvLoadImage(IMG_FILENAME14,0),0,0,IMG_ROWS,IMG_COLS);
img15 = Crop(cvLoadImage(IMG_FILENAME15,0),0,0,IMG_ROWS,IMG_COLS);
img16 = Crop(cvLoadImage(IMG_FILENAME16,0),0,0,IMG_ROWS,IMG_COLS);
img17 = Crop(cvLoadImage(IMG_FILENAME17,0),0,0,IMG_ROWS,IMG_COLS);
//Check if all images loaded correctly
if((!img1)||(!img2)||(!img3)||(!img4)||(!img5)||(!img6)||(!img7)||
    (!img8)||(!img9)||(!img10)||(!img11)||(!img12)||(!img13)||
    (!img14)||(!img15)||(!img16)||(!img17))
{
    printf("Could not load an image file\n");
    exit(0);
}
//current img is imgP e.g.img2, ref img is img1 e.g img1 for img2

// CALUCULATE MOTION VECTORS
//===============================================================================
// Exhaustive search with texture filter
//motionEstES2(motionEstVct21, img2, img1);
motionEstES2(motionEstVct32, img3, img2);
motionEstES2(motionEstVct43, img4, img3);
motionEstES2(motionEstVct54, img5, img4);
motionEstES2(motionEstVct65, img6, img5);
motionEstES2(motionEstVct67, img7, img6);
motionEstES2(motionEstVct78, img8, img7);
motionEstES2(motionEstVct89, img9, img8);
motionEstES2(motionEstVct910, img9, img8);
motionEstES2(motionEstVct1011, img10, img10);
motionEstES2(motionEstVct1211, img12, img11);
motionEstES2(motionEstVct1312, img13, img12);
motionEstES2(motionEstVct1413, img14, img13);
motionEstES2(motionEstVct1514, img15, img14);
motionEstES2(motionEstVct1615, img16, img15);
motionEstES2(motionEstVct1716, img17, img16);
motionEstES2(motionEstVct1701, img17, img1);
// TEMPORAL ACCUMULATION FOR NOISE CANCELLATION
//===============================================================================

// Add motion vectors as sets A, B, C, D and total (overall)
for (i=0; i<2; i++)
{
    for (j=0; j<num_vectors; j++)
    {
        motionVectB[i][j] = motionVect65[i][j] + motionVect76[i][j] + motionVect87[i][j] + motionVect98[i][j];
        motionVectC[i][j] = motionVect109[i][j] + motionVect110[i][j] + motionVect121[i][j] + motionVect132[i][j];
        motionVectD[i][j] = motionVect1413[i][j] + motionVect1514[i][j] + motionVect1615[i][j] + motionVect1716[i][j];
    }
}

// IDENTIFY AND LOCATE OBSTACLES TO BE AVOIDED
//===============================================================================

// Extract vector components and calculate magnitudes
// Also convert 2xnum_vectors dimensions into mv_rows x mv_cols matrices
int mbCount=0;
for (i=0; i<mv_rows; i++)
{
    for (j=0; j<mv_cols; j++)
    {
        v[i][j] = motionVect1701[0][mbCount];
        u[i][j] = motionVect1701[1][mbCount];
        magnitude[i][j] = sqrt(v[i][j]*v[i][j] + u[i][j]*u[i][j]);
        vA[i][j] = motionVectA[0][mbCount];
        uA[i][j] = motionVectA[1][mbCount];
        vB[i][j] = motionVectB[0][mbCount];
        uB[i][j] = motionVectB[1][mbCount];
        magnitudeB[i][j] = sqrt(vB[i][j]*vB[i][j] + uB[i][j]*uB[i][j]);
        vC[i][j] = motionVectC[0][mbCount];
        uC[i][j] = motionVectC[1][mbCount];
        magnitudeC[i][j] = sqrt(vC[i][j]*vC[i][j] + uC[i][j]*uC[i][j]);
        vD[i][j] = motionVectD[0][mbCount];
        uD[i][j] = motionVectD[1][mbCount];
        magnitudeD[i][j] = sqrt(vD[i][j]*vD[i][j] + uD[i][j]*uD[i][j]);
        vTotal[i][j] = mvTotal[0][mbCount];
        uTotal[i][j] = mvTotal[1][mbCount];
        magnitudeTotal[i][j] = sqrt(vTotal[i][j]*vTotal[i][j] + uTotal[i][j]*uTotal[i][j]);
        mbCount++;
    }
}

// CALL FUNCTION FOR EGO-MOTION CORRECTION AND OBS DETECTION
//===============================================================================

// Define variable of struct type detectedObstacles
struct detectedObstacles detectedObstaclesA;
struct detectedObstacles detectedObstaclesB;
struct detectedObstacles detectedObstaclesC;
struct detectedObstacles detectedObstaclesD;
struct detectedObstacles detectedObstaclesTotal;

float minnormOuter=MAXCOST; // to be used for vert-lat-roll motion correction
float FourMapsAdded[3][4], Norm16VecAdded[3][4];
for (i=0; i<3; i++)
{
    for (j=0; j<4; j++)
    {
        FourMapsAdded[i][j]=0;
        Norm16VecAdded[i][j]=0;
    }
}
//Call function to find norm and obs map
detectedObstaclesA = detectObstacles(magnitudeA, mv_rows, mv_cols);
detectedObstaclesB = detectObstacles(magnitudeB, mv_rows, mv_cols);
detectedObstaclesC = detectObstacles(magnitudeC, mv_rows, mv_cols);
detectedObstaclesD = detectObstacles(magnitudeD, mv_rows, mv_cols);
detectedObstaclesTotal = detectObstacles(magTotal, mv_rows, mv_cols);

//CALCULATE AND DISPLAY THREAT MAPS (Higher numbers indicate obstacles ahead)
//=================================================================================
//Fwd motion corrected map
printf("OBSTACLE DETECTION VIA MOTION ESTIMATION");
printf("n\nLarger numbers indicate obstacles ahead\nSmaller numbers guide where to turn to avoid obstacles\n\nForward Motion Corrected Map\n=============================");
for (i=0; i<3; i++)
{
    printf("\n");
    for (j=0; j<4; j++)
    {
        FourMapsAdded[i][j]+=detectedObstaclesA.ExpObsMap[i][j]+detectedObstaclesB.ExpObsMap[i][j]+detectedObstaclesC.ExpObsMap[i][j]+detectedObstaclesD.ExpObsMap[i][j];
        printf("%8.0f", FourMapsAdded[i][j]);
    }
    if (! ((i==1 && j==1)||(i==1 && j==2)) ) //fwd front 2 blocks are avoided
        //Also calculate minnormOuter for normalization for Lat-Vert-Roll correction
        if ((detectedObstaclesTotal.Norm[i][j]<minnormOuter) && (detectedObstaclesTotal.Norm[i][j]!=0) ) //Change 5/5/10 for normalization by min
            minnormOuter=detectedObstaclesTotal.Norm[i][j];
    }

//Lat-Vert-Roll motion corrected map
//(Correction for v,w,theta,si & phi)
if (minnormOuter<=0)
    minnormOuter=1;//To avoid division by zero or negative number for normalization
printf("\n\nLat-Vert-Roll Motion Corrected Map\n====================================");
for (i=0; i<3; i++)
{
    printf("\n");
    for (j=0; j<4; j++)
    {
        Norm16VecAdded[i][j]=detectedObstaclesTotal.Norm[i][j]/minnormOuter;
        printf("%8.0f", Norm16VecAdded[i][j]);
    }
    
    }

//Correction own-ship Long-Vert-Lat motion
printf("\n\nLong-Lat-Vert Motion Corrected Map\n====================================");
for (i=0; i<3; i++)
{
    printf("\n");

    
    
    
    
}
for (j=0; j<4; j++)
    printf("%8.0f", detectedObstaclesTotal.ExpObsMap[i][j]);
}
printf("\n\n");

//DISPLAY IMAGES AND VECTORS OVERLAYED ON IMAGES
//===============================================================================
cvNamedWindow("UncorrectedMotionVectors", CV_WINDOW_AUTOSIZE);
cvShowImage("UncorrectedMotionVectors", img1);
cvWaitKey(0);
cvNamedWindow("5thImageCorrected", CV_WINDOW_AUTOSIZE);
cvShowImage("5thImageCorrected", img5);
cvWaitKey(0);
cvNamedWindow("9thImageCorrected", CV_WINDOW_AUTOSIZE);
cvShowImage("9thImageCorrected", img9);
cvWaitKey(0);
cvNamedWindow("13thImageCorrected", CV_WINDOW_AUTOSIZE);
cvShowImage("13thImageCorrected", img13);
cvWaitKey(0);
cvNamedWindow("17thImageCorrected", CV_WINDOW_AUTOSIZE);
cvShowImage("17thImageCorrected", img17);
cvWaitKey(0);
cvNamedWindow("Total", CV_WINDOW_AUTOSIZE);
cvShowImage("Total", img17);
cvWaitKey(0);

//CLEAN UP MEMORY
//===============================================================================
for (i=0;i<firstdim;i++)
{
   //cycle through all the rows
   delete [] vectors[i]; //for each row, there are 5 columns
   delete [] motionVect21[i];
   delete [] motionVect32[i];
   delete [] motionVect43[i];
   delete [] motionVect54[i];
   delete [] motionVect65[i];
   delete [] motionVect76[i];
   delete [] motionVect87[i];
   delete [] motionVect98[i];
   delete [] motionVect109[i];
   delete [] motionVect1109[i];
   delete [] motionVect1211[i];
   delete [] motionVect1312[i];
   delete [] motionVect1413[i];
   delete [] motionVect1514[i];
   delete [] motionVect1615[i];
   delete [] motionVect1716[i];
   delete [] motionVect1701[i];
   delete [] motionVectA[i];
   delete [] motionVectB[i];
   delete [] motionVectC[i];
   delete [] motionVectD[i];
   delete [] mvTotal[i];
}
for (i=0;i<mv_rows;i++)
{
   delete [] v[i];
   delete [] u[i];
   delete [] magnitude[i];
   delete [] vA[i];
   delete [] uA[i];
   delete [] magnitudeA[i];
   delete [] vB[i];
   delete [] uB[i];
   delete [] magnitudeB[i];
   delete [] vC[i];
   delete []uC[i];
   delete [] magnitudeC[i];
   delete [] vD[i];
   delete []uD[i];
   delete [] magnitudeD[i];
}
delete [] vTotal[i];
delete [] uTotal[i];
delete [] magTotal[i];
}
delete [] vectors;
delete [] motionVect21;
delete [] motionVect32;
delete [] motionVect43;
delete [] motionVect54;
delete [] motionVect65;
delete [] motionVect76;
delete [] motionVect87;
delete [] motionVect98;
delete [] motionVect109;
delete [] motionVect1110;
delete [] motionVect1211;
delete [] motionVect1312;
delete [] motionVect1413;
delete [] motionVect1514;
delete [] motionVect1615;
delete [] motionVect1716;
delete [] motionVect1701;
delete [] motionVectA;
delete [] motionVectB;
delete [] motionVectC;
delete [] motionVectD;
delete [] mvTotal;
delete [] v;
delete [] u;
delete [] magnitude;
delete [] vA;
delete [] uA;
delete [] magnitudeA;
delete [] vB;
delete [] uB;
delete [] magnitudeB;
delete [] vC;
delete [] uC;
delete [] magnitudeC;
delete [] vD;
delete []uD;
delete [] magnitudeD;
delete [] vTotal;
delete [] uTotal;
delete [] magTotal;
cvReleaseImage(&img1); cvReleaseImage(&img2); cvReleaseImage(&img3);
cvReleaseImage(&img4); cvReleaseImage(&img5); cvReleaseImage(&img6);
cvReleaseImage(&img7); cvReleaseImage(&img8); cvReleaseImage(&img9);
cvReleaseImage(&img10); cvReleaseImage(&img11); cvReleaseImage(&img12);
cvReleaseImage(&img13); cvReleaseImage(&img14); cvReleaseImage(&img15);
cvReleaseImage(&img16); cvReleaseImage(&img17);
cvDestroyWindow("UncorrectedMotionVectors");
cvDestroyWindow("5thImageCorrected"); cvDestroyWindow("9thImageCorrected");
cvDestroyWindow("13thImageCorrected"); cvDestroyWindow("17thImageCorrected");
cvDestroyWindow("Total");
} //main function ends here

//****************************************************************************
//                            FUNCTIONS CALLED BY MAIN
//****************************************************************************

.nasa
// Function to Crop all images to known IMG_ROWSxIMG_COLS size

IplImage* Crop(IplImage* src, int startrow, int startcol, int rowwidth, int colwidth)
{
    //Dimensions of output image
    //int x,y;
    cvSetImageROI(src, cvRect(startcol, startrow, colwidth, rowwidth));
    IplImage* cropped = cvCreateImage(cvGetSize(src), src->depth, 1);
    //What the source region of interest (crop dimensions) is
    //Copy from source to cropped
    cvCopy(src, cropped);
    cvResetImageROI(src);
    return cropped;
}

void CropRelease(IplImage* src)
{
    cvReleaseImage(&src);
}

//********************************************************************************
// Exhaustive Search Motion Estimation:
// Calculates motion vectors between 2 images
// Uses noise suppression texture filter
//********************************************************************************

void motionEstES2(float** vectors, IplImage* imgP,IplImage* imgI)
{
    minCostfound testminCostfound;
    float** costs=new float*[2*PPP+1];
    static int faujiError=0;
    int i, j, mbCount, r, s;
    for (i=0;i<2*PPP+1;i++)
        costs[i] = new float[2*PPP+1];
    double mean;
    double stddev;
    IplImage* mask=0;
    IplImage *blkP;  //image block
    IplImage *blkI;  //image block
    float variance=0.0;
    float** BlockP = new float*[MBSIZE];
    float** BlockI = new float*[MBSIZE];
    for (i=0;i<MBSIZE;i++)
    {
        BlockP[i] = new float[MBSIZE];
        BlockI[i] = new float[MBSIZE];
    }
    int startrowP, startcolP, startrowI, startcolI;
    int height, width, step, channels;
    unsigned char *dataP, *dataI;
    //initialize cost matrix & motionvectors
    for (i=0; i<2*PPP+1; i++)
    {
        for (j=0; j<2*PPP+1; j++)
            costs[i][j]=MAXCOST;
    }
for (i=0; i<2; i++)
{
    for (j=0; j<num_vectors; j++)
        vectors[i][j]=0.0;
}

// Exhaustive search linearly in the search window
// We will find cost for 2p+1 x 2p+1 block vertically & horizontally
// m is row(vertical) index, n is col(horizontal) index
// we are scanning in raster order
//====================================================================================

mbCount=0;
for (i=0; i<(IMG_ROWS-MBSIZE+1); i+=MBSIZE)
{
    printf("i=%d\n", i);
    blkP=0;
    for (j=0; j<(IMG_COLS-MBSIZE+1); j+=MBSIZE)
    {
        printf("i=%d j=%d\n", i, j);

        //TEXTURE FILTER on current blk
        //Crop out current block from current imgP
        startrowP=i;
        startcolP=j;
        blkP=Crop(imgP,startrowP,startcolP,MBSIZE, MBSIZE);
        if (blkP==0)
        {
            faijiError++;
            continue;
        }

        //now check variance of this block matrix
        cvMean_StdDev(blkP, &mean, &stddev, mask);
        variance=stddev*stddev;

        if (variance<VARTH)
        {
            if (mbCount>1199)
            {
                faijiError++;
                vectors[0][mbCount]=0;
                vectors[1][mbCount]=0;
                mbCount=mbCount+1;
                continue;
            }
        }

        //CALCULATE MOTION VECTORS FOR ALL OTHER BLOCKS AFTER TEXTURE FILTER
        //=======================================================================
        for(int m=-PPP; m<=PPP; m++)
        {
            blkI=0;
            for (int n=-PPP; n<=PPP; n++)
            {
                if (i==120 &amp; j==32)
                {
                    printf("m=%d n=%d\n", m, n);
                }

                if (i==120 &amp; j==32 &amp; m==9 &amp; n==3)
                {
                    faijiError=1;
                }

                int refBlkVer = i+m; // row/vert coordinate of ref block
                int refBlkHor = j+n; // col/hor coordinate of it
                if (refBlkVer < 0 || (refBlkVer+MBSIZE) > IMG_ROWS
                    || refBlkHor < 0 || (refBlkHor+MBSIZE) > IMG_COLS)
                    continue;

                //Define block from ref image I
                startrowI=refBlkVer;
                startcolI=refBlkHor;

                blkI=Crop(imgI,startrowI,startcolI,MBSIZE, MBSIZE);
                if (blkI==0) 
                {

                    140
faujiError++;
// continue;
}
// Extract image data from both blocks in imgP and imgI
dataP = (unsigned char *)blkP->imageData;
height = MBSIZE/blkP->height;
width = MBSIZE/blkP->width;
step = blkP->widthStep;
channels = blkP->nChannels;
dataI = (unsigned char *)blkI->imageData;
// Convert image blocks to float data types
for(int k=0;k<height;k++)
{
    for(int l=0;l<width;l++)
    {
        BlockP[k][l] = (float)dataP[k*step+l*channels+0];
        BlockI[k][l] = (float)dataI[k*step+l*channels+0];
    }
}
//Calculate cost matrix between these data blocks
costs[(int)(m+PPP)][(int)(n+PPP)] = costFuncMAD(BlockP, BlockI, MBSIZE);
// if (blkI)
// CropRelease(blkI);
//
// FIND VECTORS WHERE COST IS MINIMUM
// and store it ... this is what will be passed back.
//=================================================
testminCostfound = minCost(costs);
if(mbCount>1199)
    faujiError++;

vectors[0][mbCount] = testminCostfound.dy-PPP;//-1row coordinate for vector
vectors[1][mbCount] = testminCostfound.dx-PPP;//-1col coordinate for vector
mbCount++;
//initialize cost matrix for next loop
for (r=0; r<2*PPP+1; r++)
{
    for (s=0; s<2*PPP+1; s++)
    {
        costs[r][s] = MAXCOST;
    }
}
// end of j for loop for cols
if (blkP)
    CropRelease(blkP);
// end of i for loop for rows
for (i=0;i<2*PPP+1;i++)
delete [] costs[i];
for (i=0;i<MBSIZE;i++)
{
    delete [] BlockP[i];
    delete [] BlockI[i];
}
delete [] costs;
delete [] BlockP;
delete [] BlockI;
}

//********************************************************************************
//                               costFuncMAD function
//               calculates cost as mean abs diff b/w 2 blocks
float costFuncMAD(float** currentBlk, float** refBlk, int q)
//inputs are image blocks converted to data type float
{
    int i, j;
    float err=0.0;
    float cost=0.0;
    float diff=0.0;
    for (i=0; i<q; i++)
    {
        for (j=0; j<q; j++)
        {
            diff=currentBlk[i][j] - refBlk[i][j];
            if (diff<0.0)
                diff=-diff;
            err=err+diff;
        }
    }
    cost=err/(q*q);
    return cost;
}

//*************************************************************************************/
//                                    minCost function
//                Searches for min cost in the Costs matrix b/w 2 images
//            and returns the min cost value and its location in the matrix
//*************************************************************************************/

minCostFound minCost(float **costs)
{
    minCostFound testminCostFound;
    int i, j;
    float minimum=MAXCOST;
    int row=0, col=0;
    for (i=0; i<2*PPP+1; i++)
    {
        for (j=0; j<2*PPP+1; j++)
        {
            if (costs[i][j]< minimum)
            {
                minimum=costs[i][j];
                col=j; row=i;
            }
        }
    }
    testminCostFound.dx=col;
    testminCostFound.dy=row;
    testminCostFound.min=minimum;
    return testminCostFound;
}

//*************************************************************************************/
//                            detectObstacles function
//                Does correction for all ego-motion and generates
//         obstacle avoidance maps for forward and vert-lat motion correction
//*************************************************************************************/
detectedObstacles detectObstacles(float **magnitude, int mv_rows, int mv_cols) {
    int i, j;
    int m=0;
    int n=0;
    float fwdweightage=1.25; // Change 05/05/10: More weightage given to motion exactly ahead
    //Divide magnitude matrix into mega blocks (sections)
    //Magnitude matrix is mv_rows x mv_cols size
    int section_rows=mv_rows/3;
    int section_cols=mv_cols/4; //3x4 sections, each of section_rows x section_cols
    detectedObstacles testDetectedObstacles;

    float** section11 = new float*[section_rows];  //row allocations
    float** section12 = new float*[section_rows];
    float** section13 = new float*[section_rows];
    float** section14 = new float*[section_rows];
    float** section21 = new float*[section_rows];
    float** section22 = new float*[section_rows];
    float** section23 = new float*[section_rows];
    float** section24 = new float*[section_rows];
    float** section31 = new float*[section_rows];
    float** section32 = new float*[section_rows];
    float** section33 = new float*[section_rows];
    float** section34 = new float*[section_rows];

    float** flipsection14 = new float*[section_rows];
    float** flipsection24 = new float*[section_rows];
    float** flipsection34 = new float*[section_rows];
    float** flipsection13 = new float*[section_rows];
    float** flipsection23 = new float*[section_rows];
    float** flipsection33 = new float*[section_rows];

    for (i=0; i<section_rows; i++)
    {
        section11[i] = new float[section_cols];
        section12[i] = new float[section_cols];
        section13[i] = new float[section_cols];
        section14[i] = new float[section_cols];
        section21[i] = new float[section_cols];
        section22[i] = new float[section_cols];
        section23[i] = new float[section_cols];
        section24[i] = new float[section_cols];
        section31[i] = new float[section_cols];
        section32[i] = new float[section_cols];
        section33[i] = new float[section_cols];
        section34[i] = new float[section_cols];

        flipsection14[i] = new float[section_cols];
        flipsection24[i] = new float[section_cols];
        flipsection34[i] = new float[section_cols];
        flipsection13[i] = new float[section_cols];
        flipsection23[i] = new float[section_cols];
        flipsection33[i] = new float[section_cols];
    }

    float sumnorm11 = 0, norm11 = 0;
    float sumnorm12 = 0, norm12 = 0;
    float sumnorm13 = 0, norm13 = 0;
    float sumnorm14 = 0, norm14 = 0;
    float sumnorm21 = 0, norm21 = 0;
    float sumnorm22 = 0, norm22 = 0;
    float sumnorm23 = 0, norm23 = 0;
    float sumnorm24 = 0, norm24 = 0;
    float sumnorm31 = 0, norm31 = 0;
    float sumnorm32 = 0, norm32 = 0;
    float sumnorm33 = 0, norm33 = 0;
    float sumnorm34 = 0, norm34 = 0;
    float SAD31_34 = 0, SAD11_14 = 0;
    float SAD32_33 = 0, SAD12_13 = 0;
    float SAD21_24 = 0, SAD22_23 = 0;
    float meanOuter = 0, stddevOuter = 0;
    float Threshold1, Threshold2, Threshold3, meanInner;
    float minSAD, SADthreshold;
    int flag11 = 0, flag12 = 0, flag13 = 0, flag14 = 0;
}

143
int flag21=0, flag22=0, flag23=0, flag24=0;
int flag31=0, flag32=0, flag33=0, flag34=0;

//SEPARATE MOTION VECTOR MAGNITUDE MATRIX INTO 12 SECTIONS
//Such separation is spatial accumulation for noise handling
for (i=0; i<mv_rows; i++)
{
    for (j=0; j<mv_cols; j++)
    {
        if (i<section_rows)
        {
            if (j<section_cols)
                section11[m][n]=magnitude[0][0];
            else if (j<2*section_cols)
                section12[m][n]=magnitude[i][j];
            else if (j<3*section_cols)
                section13[m][n]=magnitude[i][j];
            else
                section14[m][n]=magnitude[i][j];
        }
        else if (i<2*section_rows)
        {
            if (j<section_cols)
                section21[m][n]=magnitude[i][j];
            else if (j<2*section_cols)
                section22[m][n]=magnitude[i][j];
            else if (j<3*section_cols)
                section23[m][n]=magnitude[i][j];
            else
                section24[m][n]=magnitude[i][j];
        }
        else
        {
            if (j<section_cols)
                section31[m][n]=magnitude[i][j];
            else if (j<2*section_cols)
                section32[m][n]=magnitude[i][j];
            else if (j<3*section_cols)
                section33[m][n]=magnitude[i][j];
            else
                section34[m][n]=magnitude[i][j];
        }
        n++;
        if (n==section_cols)
        {
            n=0;
        }
    }
    m++;
    if (m==section_rows)
    {
        m=0;
    }
}

//OWN-SHIP MOTION CORRECTION WITH SPATIAL ACCUMULATION
//Calculate sum of absolute differences with flipped dimension for fwd correction
//Calculate norm matrices for lat-vert motion correction
for (m=0; m<section_rows; m++)
{
    for (n=0; n<section_cols; n++)
    {
        int k=section_cols-1;
        sumnorm11+=section11[m][n]*section11[m][n];
        sumnorm12+=section12[m][n]*section12[m][n];
        sumnorm13+=section13[m][n]*section13[m][n];
        sumnorm14+=section14[m][n]*section14[m][n];
        sumnorm21+=section21[m][n]*section21[m][n];
        sumnorm22+=section22[m][n]*section22[m][n];
        sumnorm23+=section23[m][n]*section23[m][n];
        sumnorm24+=section24[m][n]*section24[m][n];
        sumnorm31+=section31[m][n]*section31[m][n];
        sumnorm32+=section32[m][n]*section32[m][n];
        sumnorm33+=section33[m][n]*section33[m][n];
    }
}
sumnorm34+=section34[m][n]*section34[m][n];

//flip (mirror) right half sections for comparison with left half
flipsection14[m][k]=section14[m][n];
flipsection24[m][k]=section24[m][n];
flipsection34[m][k]=section34[m][n];
flipsection13[m][k]=section13[m][n];
flipsection23[m][k]=section23[m][n];
flipsection33[m][k]=section33[m][n];
k--;
}

SAD31_34=costFuncMAD(section31, flipsection34, section_rows);
SAD11_14=costFuncMAD(section11, flipsection14, section_rows);
SAD32_33=costFuncMAD(section32, flipsection33, section_rows);
SAD12_13=costFuncMAD(section12, flipsection13, section_rows);
SAD21_24=costFuncMAD(section21, flipsection24, section_rows);
SAD22_23=costFuncMAD(section22, flipsection23, section_rows);

norm11=sqrt(sumnorm11); norm12=sqrt(sumnorm12);
norm13=sqrt(sumnorm13); norm14=sqrt(sumnorm14);
norm21=sqrt(sumnorm21); norm22=sqrt(sumnorm22);
norm23=sqrt(sumnorm23); norm24=sqrt(sumnorm24);
norm31=sqrt(sumnorm31); norm32=sqrt(sumnorm32);
norm33=sqrt(sumnorm33); norm34=sqrt(sumnorm34);

//Vert-Lat motion correction only involves norms
//so we are done for this part.
//For fwd motion correction following is done:-

meanOuter=(norm11+norm12+norm13+norm14+norm21+norm24+norm31+norm32+norm33+norm34)/10;
Threshold1=meanOuter+stddevOuter;
Threshold2=Threshold1+stddevOuter;
Threshold3=Threshold2+stddevOuter;
meanInner=meanOuter-stddevOuter;

//Another threshold from SADs calculated above
minSAD=MAXCOST;
if (SAD31_34<minSAD)
minSAD=SAD31_34;
if (SAD11_14<minSAD)
minSAD=SAD11_14;
if (SAD32_33<minSAD)
minSAD=SAD32_33;
if (SAD12_13<minSAD)
minSAD=SAD12_13;
if (SAD21_24<minSAD)
minSAD=SAD21_24;
if (SAD22_23<minSAD)
minSAD=SAD22_23;
if (minSAD<0.01)
minSAD=0.01; //Avoid divide by 0 or negative value
SADthreshold=minSAD;

//DETECTING OBSTACLES USING MIRROR COMPARISONS FOR OWN-SHIP FWD MOTION CORRECTION
//-------------------------------------------------------------------------------------

//Compare section 31, 34
if ((SAD31_34-SADthreshold)/SADthreshold<=DELTA)
  //printf("\nNo obstacles expected in sections 31, 34")
else
  {
    if (norm31>=meanOuter)
      flag31=1;
    if (norm31>=Threshold1)
      flag31=4;
    if (norm31>=Threshold2)
      flag31=6;
    if (norm31>=Threshold3)
      flag31=8;
  }
if (norm34>=meanOuter) flag34=1;
if (norm34>=Threshold1) flag34=4;
if (norm34>=Threshold2) flag34=6;
if (norm34>=Threshold3) flag34=8;
}

//Compare section 11, 14
if ( (SAD11_14-SADthreshold)/SADthreshold <= DELTA )
//printf("No obstacles expected in sections 11, 14")
{
    //Both section 11, 14 are almost exactly same, flags remain 0 as initialized
}
else
{
    if (norm11>=meanOuter) flag11=1;
    if (norm11>=Threshold1) flag11=4;
    if (norm11>=Threshold2) flag11=6;
    if (norm11>=Threshold3) flag11=8;
    if (norm14>=meanOuter) flag14=1;
    if (norm14>=Threshold1) flag14=4;
    if (norm14>=Threshold2) flag14=6;
    if (norm14>=Threshold3) flag14=8;
}

//Compare section 32, 32
if ( (SAD32_33-SADthreshold)/SADthreshold<=DELTA)
//printf("No obstacles expected in sections 32, 32")
{
    //Both section 32, 32 are almost exactly same, flags remain 0 as initialized
}
else
{
    if (norm32>=meanOuter) flag32=1;
    if (norm32>=Threshold1) flag32=4;
    if (norm32>=Threshold2) flag32=6;
    if (norm32>=Threshold3) flag32=8;
    if (norm33>=meanOuter) flag33=1;
    if (norm33>=Threshold1) flag33=4;
    if (norm33>=Threshold2) flag33=6;
    if (norm33>=Threshold3) flag33=8;
}

//Compare section 21, 24
if ( (SAD21_24-SADthreshold)/SADthreshold<=DELTA)
//printf("No obstacles expected in sections 21, 24")
{
    //Both section 21, 24 are almost exactly same, flags remain 0 as initialized
}
else
{
    if (norm21>=meanOuter) flag21=1;
    if (norm21>=Threshold1) flag21=4;
    if (norm21>=Threshold2) flag21=6;
    if (norm21>=Threshold3) flag21=8;
    if (norm24>=meanOuter) flag24=1;
    if (norm24>=Threshold1) flag24=4;
    if (norm24>=Threshold2) flag24=6;
    if (norm24>=Threshold3) flag24=8;
}
//Compare section 12, 13
if ((SAD12_13-SADthreshold)/SADthreshold<=DELTA)
    //printf("no obstacles expected in sections 12, 13")
else
    
        if (norm12>=meanOuter)
            flag12=1;
        if (norm12>=Threshold1)
            flag12=4;
        if (norm12>=Threshold2)
            flag12=6;
        if (norm12>=Threshold3)
            flag12=8;
        if (norm13>=meanOuter)
            flag13=1;
        if (norm13>=Threshold1)
            flag13=4;
        if (norm13>=Threshold2)
            flag13=6;
        if (norm13>=Threshold3)
            flag13=8;

    //Compare section 22, 23
    //As this is directly in front, a little more stringent criteria is used
    if ((SAD22_23-SADthreshold)/SADthreshold<=DELTA)
        //printf("no obstacles expected in sections 22, 23")
    else
        
            if (norm22>=meanInner)
                flag22=1;
            if (norm22>=meanOuter)
                flag22=3;
            if (norm22>=Threshold1)
                flag22=5;
            if (norm22>=Threshold2)
                flag22=7;
            if (norm23>=meanInner)
                flag23=1;
            if (norm23>=meanOuter)
                flag23=3;
            if (norm23>=Threshold1)
                flag23=5;
            if (norm23>=Threshold2)
                flag23=7;

    //OUTPUT THE RESULTS
   =================================================================================
    //Norm is for vert-lat motion correction and ExpObsMap for fwd
testDetectedObstacles.Norm[0][0]=norm11; testDetectedObstacles.Norm[0][1]=norm12;
testDetectedObstacles.Norm[0][2]=norm13; testDetectedObstacles.Norm[0][3]=norm14;
testDetectedObstacles.Norm[1][0]=norm21; testDetectedObstacles.Norm[1][1]=norm22*fwdweightage;
testDetectedObstacles.Norm[1][2]=norm23*fwdweightage; //change 5/5/10, more wtg to motion in front
testDetectedObstacles.Norm[1][3]=norm24;
testDetectedObstacles.Norm[2][0]=norm31; testDetectedObstacles.Norm[2][1]=norm32;
testDetectedObstacles.ExpObsMap[0][0]=flag11;testDetectedObstacles.ExpObsMap[0][1]=flag12;
testDetectedObstacles.ExpObsMap[0][2]=flag13; testDetectedObstacles.ExpObsMap[0][3]=flag14;
testDetectedObstacles.ExpObsMap[1][0]=flag21;testDetectedObstacles.ExpObsMap[1][1]=flag22;
testDetectedObstacles.ExpObsMap[2][0]=flag31; testDetectedObstacles.ExpObsMap[2][1]=flag32;
return testDetectedObstacles;

//MEMORY CLEAN UP
=================================================================================
for (i=0;i<section_rows;i++)
}\n\n}; //function detectObstacle ends here

//END OF CODE
Published or accepted contributions based on the work presented in this dissertation are as follows [76-81]:


REFERENCES


