On-board three-dimensional object tracking: Software and hardware solutions

Ajay Kumar Mandava
University of Nevada Las Vegas

Follow this and additional works at: http://digitalscholarship.unlv.edu/thesesdissertations

Part of the Computer Engineering Commons, Electrical and Computer Engineering Commons, and the Military and Veterans Studies Commons

Repository Citation
http://digitalscholarship.unlv.edu/thesesdissertations/169
ON-BOARD 3D OBJECT TRACKING: SOFTWARE AND HARDWARE

SOLUTIONS

by

Ajay Kumar Mandava

Bachelor of Technology in Electronics and Communication Engineering
Jawaharlal Nehru Technological University, India
May 2006

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Electrical Engineering
Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2009
THE GRADUATE COLLEGE

We recommend that the thesis prepared under our supervision by

**Ajay Kumar Mandava**

entitled

**On-Board 3D Object Tracking: Software and Hardware Solutions**

be accepted in partial fulfillment of the requirements for the degree of

**Master of Science**
Electrical Engineering

Emma E. Regentova, Committee Chair
Muthukumar Venkatesan, Committee Member
Henry Selvaraj, Committee Member
Ajoy Kumar Datta, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2009
ABSTRACT

On-Board 3D Object Tracking: Software and Hardware Solutions

by

Ajay Kumar Mandava

Dr. Emma E. Regentova, Examination Committee Chair
Associate Professor of Electrical and Computer Engineering,
University of Nevada, Las Vegas

We describe a real time system for recognition and tracking 3D objects such as UAVs, airplanes, fighters with the optical sensor. Given a 2D image, the system has to perform background subtraction, recognize relative rotation, scale and translation of the object to sustain a prescribed topology of the fleet. In the thesis a comparative study of different algorithms and performance evaluation is carried out based on time and accuracy constraints. For background subtraction task we evaluate frame differencing, approximate median filter, mixture of Gaussians and propose classification based on neural network methods. For object detection we analyze the performance of invariant moments, scale invariant feature transform and affine scale invariant feature transform methods. Various tracking algorithms such as mean shift with variable and a fixed sized windows, scale invariant
feature transform, Harris and fast full search based on fast fourier transform algorithms are evaluated. We develop an algorithm for the relative rotations and the scale change calculation based on Zernike moments. Based on the design criteria the selection is made for on-board implementation. The candidate techniques have been implemented on the Texas Instrument TMS320DM642 EVM board. It is shown in the thesis that 14 frames per second can be processed; that supports the real time implementation of the tracking system under reasonable accuracy limits.
TABLE OF CONTENTS

ABSTRACT .................................................................................................................. iii

LIST OF FIGURES .................................................................................................. viii

LIST OF TABLES .................................................................................................... x

ACKNOWLEDGEMENTS ...................................................................................... xi

CHAPTER 1 INTRODUCTION ................................................................................. 1
  1.1 Overview of the Project ................................................................................. 1
  1.2 Outline of Thesis ........................................................................................ 5

CHAPTER 2 BACKGROUND RESEARCH .......................................................... 7
  2.1 Background Subtraction ............................................................................ 7
  2.2 Object Detection/ Identification ............................................................. 12
    2.2.1 Geometry Based Approaches ....................................................... 13
    2.2.2 Appearance Based Algorithms ................................................... 13
    2.2.3 Feature Based Algorithms ......................................................... 15
  2.3 Object Tracking ....................................................................................... 16
    2.3.1 Point Tracking ............................................................................. 16
    2.3.2 Kernel Tracking ......................................................................... 18
    2.5.3 Silhouette Tracking ....................................................................... 19
  2.4 Pose Estimation ....................................................................................... 20

CHAPTER 3 SEGMENTATION/ BACKGROUND SUBTRACTION .................... 25
  3.1 Moving Object Segmentation Methods .................................................. 25
    3.1.1 Frame Differencing ...................................................................... 25
    3.1.2 Approximated Median Filter ........................................................ 28
    3.1.3 Mixture of Gaussians ................................................................... 31
    3.1.4 Feature Calculation and Classification Based on Neural Network ... 33

CHAPTER 4 OBJECT TRACKING ......................................................................... 49
  4.1 Mean Shift Tracking ................................................................................. 49
    4.1.1 Implementation of the Mean Shift Tracking ................................... 51
    4.1.2 Mean Shift Tracking-1 ................................................................... 51
    4.1.3 Mean Shift Tracking-2 ................................................................... 55
4.2 Scale Invariant Feature Transform Based Tracking .......... 59
   4.2.1 Algorithm Description ........................................ 60
4.3 Harris Based Tracking .............................................. 62
   4.3.1 Harris Corner Detector ...................................... 62
   4.3.2 Point Matching .............................................. 63
   4.3.3 Computing Fundamental Matrix and Essential Matrix .................................................... 65
   4.3.4 RANSAC .................................................... 66
4.4 Fast Full Search Based on FFT Algorithms ................. 68
   4.4.1 Algorithm Description ...................................... 70

CHAPTER 5 OBJECT DETECTION AND POSE ESTIMATION ......... 75
5.1 Moment Functions for Object Identification ................. 75
   5.1.1 Moment Based Method ...................................... 75
   5.1.2 Complex Radial Moments .................................. 78
   5.1.3 Zernike Moments .......................................... 79
5.2 Scale Invariant Feature Transform Based Object Detection ...... 80
   5.2.1 Construction of a Scale Space Representation ........ 81
   5.2.2 Local Extrema Detection ................................... 82
   5.2.3 Removal of Unstable Keypoints Along the Edges ....... 84
   5.2.4 Orientation Assignment ................................... 86
   5.2.5 Keypoint Descriptor Construction ...................... 88
   5.2.6 Object Recognition ....................................... 90
   5.2.7 Keypoint Matching ....................................... 90
   5.2.8 Nearest Neighbors ....................................... 91
   5.2.9 Object Matching ....................................... 92
   5.2.10 Recognition Examples .................................... 92
5.3 Affine SIFT .......................................................... 99
   5.3.1 When does it work? ....................................... 99
   5.3.2 The ASIFT Algorithm ................................... 100
   5.3.3 Recognition Examples .................................. 103
5.4 Object Detection .................................................. 108
5.5 Moment Functions for Pose Identification .................. 123

CHAPTER 6 HARDWARE IMPLEMENTATION ............................ 131
6.0 Introduction ....................................................... 131
6.1 Embedded MATLAB ............................................... 131
6.2 TMS320DM642 processor ......................................... 135
   6.2.1 C64x Architecture Overview ......................... 136
   6.2.2 Chip Level Features .................................... 137
   6.2.3 Code Development .................................... 143
   6.2.4 DSP/BIOS ........................................... 144
   6.2.5 Chip Support Library .................................. 147
   6.2.6 Network Developer’s Kit .............................. 148
6.3 The Experimental Setup .......................................... 149
6.4 Real Time Tracking/ Pose Estimation System ................ 150
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Cooperative flight</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>System overview</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Pitch, Roll and Yaw</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Input Frame</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Output Frame with threshold = 25</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Output Frame with threshold = 5 (low)</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>Output Frame with threshold = 50 (high)</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Input Frame</td>
<td>30</td>
</tr>
<tr>
<td>3.6</td>
<td>Output Frame for threshold</td>
<td>30</td>
</tr>
<tr>
<td>3.7</td>
<td>Input Frame</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Output Frame</td>
<td>34</td>
</tr>
<tr>
<td>3.9</td>
<td>The Haar wavelet</td>
<td>35</td>
</tr>
<tr>
<td>3.10</td>
<td>Subband images</td>
<td>39</td>
</tr>
<tr>
<td>3.11</td>
<td>Proposed neural network design</td>
<td>42</td>
</tr>
<tr>
<td>3.12</td>
<td>Background segmentation (subtraction)</td>
<td>43</td>
</tr>
<tr>
<td>3.13</td>
<td>Decomposed image</td>
<td>44</td>
</tr>
<tr>
<td>3.14</td>
<td>Results of feature calculation and classification based on NN</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Tracking through a sequence</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Tracking with MST-2</td>
<td>59</td>
</tr>
<tr>
<td>4.3</td>
<td>Tracking through the sequence with SIFT</td>
<td>62</td>
</tr>
<tr>
<td>4.4</td>
<td>Tracking through the sequence with Harris</td>
<td>67</td>
</tr>
<tr>
<td>4.5</td>
<td>Search area in full search block matching algorithm</td>
<td>73</td>
</tr>
<tr>
<td>4.6</td>
<td>Tracking through the sequence with SSD FFT</td>
<td>73</td>
</tr>
<tr>
<td>5.1</td>
<td>Scale space representation example</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>Difference of gaussian</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Difference of Gaussian example</td>
<td>84</td>
</tr>
<tr>
<td>5.4</td>
<td>Selecting local extremas</td>
<td>85</td>
</tr>
<tr>
<td>5.5</td>
<td>Assigning orientations and magnitudes in orientation histogram</td>
<td>87</td>
</tr>
<tr>
<td>5.6</td>
<td>SIFT recognition examples</td>
<td>93</td>
</tr>
<tr>
<td>5.7</td>
<td>SIFT recognition examples under noise effect</td>
<td>95</td>
</tr>
<tr>
<td>5.8</td>
<td>SIFT recognition examples under occlusion</td>
<td>97</td>
</tr>
<tr>
<td>5.9</td>
<td>Overview of ASIFT</td>
<td>100</td>
</tr>
<tr>
<td>5.10</td>
<td>Sampling of the parameters $\Theta=\arccos \frac{1}{t}$ and $\Phi$ in a zenith view of the observation half sphere</td>
<td>103</td>
</tr>
<tr>
<td>5.11</td>
<td>ASIFT recognition examples</td>
<td>104</td>
</tr>
<tr>
<td>5.12</td>
<td>ASIFT recognition examples under noise effect</td>
<td>105</td>
</tr>
<tr>
<td>5.13</td>
<td>ASIFT recognition examples under occlusion</td>
<td>107</td>
</tr>
<tr>
<td>5.14</td>
<td>Database 2 objects</td>
<td>110</td>
</tr>
<tr>
<td>6.1</td>
<td>Block diagram of DM642 EVM</td>
<td>136</td>
</tr>
<tr>
<td>6.2</td>
<td>C64x CPU</td>
<td>138</td>
</tr>
<tr>
<td>6.3</td>
<td>C64x data cross paths</td>
<td>139</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>C64x memory load and store paths</td>
<td>140</td>
</tr>
<tr>
<td>Figure 6.5</td>
<td>TMS320C64x block diagram</td>
<td>140</td>
</tr>
<tr>
<td>Figure 6.6</td>
<td>TMS320C64x two level internal memory block diagram</td>
<td>141</td>
</tr>
<tr>
<td>Figure 6.7</td>
<td>Code development cycle</td>
<td>145</td>
</tr>
<tr>
<td>Figure 6.8</td>
<td>DSP/BIOS components</td>
<td>147</td>
</tr>
<tr>
<td>Figure 6.9</td>
<td>Experimental setup</td>
<td>149</td>
</tr>
<tr>
<td>Figure 6.10</td>
<td>Overview of real time tracking/pose estimation system</td>
<td>150</td>
</tr>
<tr>
<td>Figure 6.11</td>
<td>Execution graph</td>
<td>152</td>
</tr>
<tr>
<td>Figure 6.12</td>
<td>CPU load graph</td>
<td>153</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Background subtraction Classification results</td>
<td>45</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Identification accuracy (%) vs different type of distortions and scale</td>
<td>119</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Identification accuracy (%) vs scale change</td>
<td>120</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Identification accuracy (%) vs Gaussian noise</td>
<td>121</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Identification accuracy (%) vs Salt and Pepper noise</td>
<td>122</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Angle calculation from moment functions</td>
<td>124</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Average accuracy (%) under different distortions</td>
<td>127</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Accuracy (%) vs scale</td>
<td>128</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Accuracy (%) vs Gaussian noise</td>
<td>128</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>Accuracy (%) vs Poisson noise</td>
<td>129</td>
</tr>
<tr>
<td>Table 5.10</td>
<td>Accuracy (%) vs speckle noise</td>
<td>129</td>
</tr>
<tr>
<td>Table 5.11</td>
<td>Accuracy (%) vs occlusion</td>
<td>129</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Execution times comparison</td>
<td>154</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I am extremely happy to take this opportunity to acknowledge my debts and gratitude to those who were associated with the preparation of this thesis. Words fail to express my profound regards from the inmost recess of my heart to my advisor Dr. Emma E. Regentova for the invaluable help, constant guidance and wide counseling extended to me right from the selection of the research topic to the successful completion of this thesis. Also, I would like to thank Dr. Muthukumar Venkatesan for assisting and advising me throughout the research.

I am extending my sincere thanks to Dr. Henry Selvaraj, and Dr. Ajoy K. Datta, for their direct and indirect contribution throughout this thesis. Finally, I would like to thank my parents and friends for their constant encouragement.
CHAPTER 1

INTRODUCTION

1.1. Overview of the Project

This thesis is on detecting a flying object and distinguishing a change of 3D rotational angles under different scales and translations from 2D images only. Specifically, we consider a situation when other sensors malfunction or jammed and/or the communication is lost or compromised. The problem also is related to a selection of a minimum set of sensors to be used as a small flying objects such as UAV, possess a limited payload capability. We believe that although the visual data cannot remedy completely the “loss of control” situation, but it delivers the useful information for sustaining the coordinated fleet for a certain period of time.

To formulate more specifically, the problem we would like to solve is to provide a “slave” object with the information about relative change of the path by the leading object which can be an aircraft, a fighter or another UAV in a coordinated (topology and path) flight so the former recalculates its own flight parameters and sustain a prescribed topology of the group.

In Fig.1.1, we can assume a front fighter is a leader for the next in the row, and so forth. Each of the slave objects are assumed to see the leader within the same location and at the same pose in the image acquired by
the on-board camera. These to be sustained during the flight by adjusting the flight parameters accordingly in response to the leader's path and speed change. The latter is observed as a scale change in the optical image.

Fig.1.1. Cooperative flight

The overall central procedure can be explained as a sequence of steps as in Fig.1.2. We detect the image in predetermined position, track the object, estimate the pose, adjust flight parameter to be aligned. If the object is lost, redetect it and continue in the loop.
An aircraft/ UAV has three different kinds of rotations, namely Pitch, Yaw, and Roll as illustrated below in Fig.1.3. The yaw axis is defined to be perpendicular to the plane of the wings with its origin at the center of gravity and directed towards the bottom of the aircraft. A yaw motion is a movement of the nose of the aircraft from side to side. The pitch axis is perpendicular to the yaw axis and is parallel to the plane of the wings with its origin at the center of gravity and directed towards the right wing tip. A pitch motion is an up or down movement of the nose of the aircraft. The roll axis is perpendicular to the other two axes with its origin at the center of gravity, and is directed towards the nose of the aircraft. A rolling motion is an up and down movement of the wing tips of the aircraft.
There are a number of problems to solve. One is the detection of a specific object within the acquired image and tracking it. The second task is to determine 3D pose continuously or within specified intervals for establishing relative rotations, translation and scale. The pose is estimated with respect to the coordinates of the camera system and thereafter is to be recalculated with respect to the coordinate system of the flying object. The control parameters are derived accordingly but this task is the out of scope of this project. The third task is to implement the algorithm and the code in the hardware and assess the speed and the
power consumption for evaluating its possible implementation on UAV of
different scales, i.e., miniature, small and medium size.

Object identification and pose estimation procedure can be divided
into 5 steps:

1. image acquisition,
2. image processing,
3. feature extraction,
4. pose calculation.

In the first step, the image is acquired by using the camera mounted
on the UAV following the leader UAV. In the step two, background
subtraction and binarization are performed. In the third step, a set of
moment variant and invariant features is extracted from the binary
image objects. The pose is estimated based on the features.

In the training phase, each library is formed from numerous views of
N reference objects. They are of a certain scale with three rotation angles
increment of 4 degrees. While in the identification phase, an object image
is tracking and pose estimation tasks are performed.

1.2. Outline of Thesis

The structure of this thesis closely follows the order in which the work
was undertaken in response to the aims as they were initially conceived.
It consists of five further chapters.

Chapter 2 briefly discusses an overview of previous work in
background subtraction, object recognition, tracking and pose
estimation.
Chapter 3 focuses on algorithms developed for background subtraction. We propose to subtract background based on classification, and performance is evaluated. The results of the proposed algorithm are presented in this chapter.

Chapter 4 discusses the different tracking algorithms to localize the target object and conclude on a possible candidate for the application.

Chapter 5 explains an approach to the object identification and pose estimation under different poses, scale and noise environments.

Chapter 6 summarizes implementation details of the enhanced software on the TI’s TMS320C6000 DSP platform and system features and discusses the results obtained from the use of proposed algorithms implemented on DSP platform on sample and real-time video sequences.

Finally, Chapter 7 summarizes the work done within the scope of this thesis and discusses the conclusions drawn from the work carried out. It also addresses the recommendations for the similar works that are intended to be done in the future.
CHAPTER 2

BACKGROUND RESEARCH

Pose estimation, in general, is a challenging problem. In its simplest form, pose can be defined as the problem of estimating the three angles of an object in the image plane as it moves around a scene. Difficulties in finding the object pose can arise due to abrupt object motion, changing appearance patterns of the object and the scene, non-rigid object structures, object-to-object and object-to-scene occlusions, and camera motion.

There are four key steps in video analysis: background subtraction, object detection/identification of moving objects, tracking of such objects from frame to frame, and pose estimation of the object. Every pose estimation method requires tracking and object detection mechanism either in every frame or when the object first appears in the video.

2.1. Background Subtraction

Background subtraction is often one of the first tasks in machine vision applications, making it a critical part of the system. The output of background subtraction is an input to a higher level process that can be, for example, tracking of an identified object. The performance of background subtraction depends mainly on the background modeling
A large number of different methods for detecting moving objects have been proposed and many different features are utilized for modeling the background. Most of the methods use only the pixel color or intensity information to make the decision.

Frame differencing [20,21] is arguably the simplest background modeling technique, frame differencing uses the video frame at time t -1 as the background model for the frame at time t. Since it uses only a single previous frame, frame differencing may not be able to identify the interior pixels of a large, uniformly-colored moving object. This is commonly known as the aperture problem.

Median filtering is one of the most commonly-used background modeling techniques [59,60,61,62]. The background estimate is defined to be the median at each pixel location of all the frames in the buffer. The
assumption is that the pixel stays in the background for more than half of the frames in the buffer. Median filtering has been extended to color by replacing the median with the medoid [60]. The complexity of computing the median is $O(L \log L)$ for each pixel.

Toyama et al. compute the current background estimate by applying a linear predictive filter on the pixels in the buffer [63]. The filter coefficients are estimated at each frame time based on the sample covariances, making this technique difficult to apply in real-time.

Unlike previous techniques that use a single background estimate at each pixel location, Elgammal et al. [64] use the entire history $I_{t-L}; I_{t-L+1}; \ldots; I_{t-1}$ to form a non-parametric estimate of the pixel density function $f(I_t = u)$:

$$f(I_t = u) = \frac{1}{L} \sum_{i=t-L}^{t-1} K(u - I_i)$$

$K(.)$ is the kernel estimator which was chosen to be Gaussian. The current pixel $I_t$ is declared as foreground if it is unlikely to come from this distribution, i.e. $f(I_t)$ is smaller than some predefined threshold. The advantage of using the full density function over a single estimate is the ability to handle multi-modal background distribution. Examples of multi-modal background include pixels from a swinging tree or near high-contrast edges where they flicker under small camera movement. The implementation in [64] uses the median of the absolute differences
between successive frames as the width of the kernel. Thus, the complexity of building the model is the same as median filtering. On the other hand, the foreground detection is more complex as it needs to compute Equation (1) for each pixel.

Due to the success of non-recursive median filtering, McFarlane and Schofield propose a simple recursive filter to estimate the median [23]. This technique has also been used in back-ground modeling for urban traffic monitoring [65]. In this scheme, the running estimate of the median is incremented by one if the input pixel is larger than the estimate, and decreased by one if smaller. This estimate eventually converges to a value for which half of the input pixels are larger than and half are smaller than this value, that is, the median.

Kalman filter is a widely-used recursive technique for tracking linear dynamical systems under Gaussian noise. Many different versions have been proposed for background modeling, differing mainly in the state spaces used for tracking. The simplest version uses only the luminance intensity [66, 67,68]. Karmann and von Brandt use both the intensity and its temporal derivative [69], while Koller, Weber, and Malik use the intensity and its spatial derivatives [70].

Unlike Kalman filter which tracks the evolution of a single Gaussian, the MoG method tracks multiple Gaussian distributions simultaneously. MoG has enjoyed tremendous popularity since it was first proposed for background modeling in [71]. Similar to the non-parametric model, MoG
maintains a density function for each pixel. Thus, it is capable of handling multi-modal background distributions. On the other hand, since MoG is parametric, the model parameters can be adaptively updated without keeping a large buffer of video frames.

Pfinder [72] uses a simple scheme, where background pixels are modeled by a single value, updated by

\[ B_t = (1 - \alpha)B_{t-1} + \alpha I_t \]

and foreground pixels are explicitly modeled by a mean and covariance, which are updated recursively. It requires an empty scene at start-up.

In [73, 74, 75], a pixel is marked as foreground if

\[ |M - I_t| > D \quad \text{or} \quad |N - I_t| > D \]

where the (per pixel) parameters M, N, and D represent the minimum, maximum, and largest interframe absolute difference observable in the background scene. These parameters are initially estimated from the first few seconds of video and are periodically updated for those parts of the scene not containing foreground objects.

The resulting foreground image is eroded to eliminate 1-pixel thick noise, then connected component labeled and small regions rejected. Finally, the remaining regions are dilated and then eroded.

In [76], the background is updated by
\[ B_{t+1} = \alpha S(I_t) + (1 - \alpha) B_t \]

at all pixels, where \( S(I_t) \) is a smoothed version of \( I_t \). Foreground pixels are identified by tracking the maxima of \( S(I_t - B_t) \), as opposed to thresholding. They use \( \alpha = [0.3, \ldots, 0.5] \) and rely on the streaking effect to help in determining correspondence between frames. They also note that \( (1 - \alpha)^t < 0.1 \) gives an indication of the number of frames \( t \) needed for the background to settle down after initialization.

In [77], color images are used because it is claimed to give better segmentation than monochrome, especially in low contrast areas, such as objects in dark shadows. The background estimate is defined to be the temporal median of the last \( N \) frames, with typical values of \( N \) ranging from 50 to 200.

Pixels are marked as foreground if

\[ \sum_{C \in R,G,B} |I_t(C) - B_t(C)| > K \sigma \]

where \( \sigma \) is an offline generated estimate of the noise standard deviation, and \( K \) is an apriori selected constant (typically 10). This method also uses template matching to help in selecting candidate matches.

2.2. Object Detection/ Identification
2.2.1 Geometry-Based Approaches

Early attempts on object recognition were focused on using geometric models of objects to account for their appearance variation due to viewpoint and illumination change. The main idea is that the geometric description of a 3D object allows the projected shape to be accurately predicated in a 2D image under projective projection, thereby facilitating recognition process using edge or boundary information (which is invariant to certain illumination change). Much attention was made to extract geometric primitives (e.g., lines, circles, etc.) that are invariant to viewpoint change [78]. Nevertheless, it has been shown that such primitives can only be reliably extracted under limited conditions (controlled variation in lighting and viewpoint with certain occlusion). An excellent review on geometry-based object recognition research by Mundy can also be found in [79].

2.2.2 Appearance-Based Algorithms

In contrast to early efforts on geometry-based object recognition works, most recent efforts have been centered on appearance-based techniques as advanced feature descriptors and pattern recognition algorithms are developed [80]. Most notably, the eigenface methods have attracted much attention as it is one of the first face recognition systems that are computationally efficient and relatively accurate [81]. The underlying idea of this approach is to compute eigenvectors from a set of vectors where each one represents one face image as a raster scan vector.
of gray-scale pixel values. Each eigenvector, dubbed as an eigenface, captures certain variance among all the vectors, and a small set of eigenvectors captures almost all the appearance variation of face images in the training set. Given a test image represented as a vector of gray-scale pixel values, its identity is determined by finding the nearest neighbor of this vector after being projected onto a subspace spanned by a set of eigenvectors. In other words, each face image can be represented by a linear combination of eigenfaces with minimum error (often in the L2 sense), and this linear combination constitutes a compact reorientation. The eigenface approach has been adopted in recognizing generic objects across different viewpoints [82] and modeling illumination variation [83]. As the goal of object recognition is to tell one object from the others, discriminative classifiers have been used to exploit the class specific information. Classifiers such as k-nearest neighbor, neural networks with radial basis function (RBF), dynamic link architecture, Fisher linear discriminant, support vector machines (SVM), sparse network of Winnows (SNoW), and boosting algorithms have been applied to recognize 3D objects from 2D images [84] [85] [86] [87] [88]. While appearance-based methods have shown promising results in object recognition under viewpoint and illumination change, they are less effective in handling occlusion. In addition, a large set of exemplars needs to be segmented from images for generative or discriminative
methods to learn the appearance characteristics. These problems are partially addressed with parts-based representation schemes.

2.2.3. Feature-Based Algorithms

The central idea of feature-based object recognition algorithms lies in finding interest points, often occurred at intensity discontinuity, that are invariant to change due to scale, illumination and affine transformation (a brief review on interest point operators can be found in [49]). The scale-invariant feature transform (SIFT) descriptor, proposed by Lowe, is arguably one of the most widely used feature representation schemes for vision applications [49]. The SIFT approach uses extrema in scale space for automatic scale selection with a pyramid of difference of Gaussian filters, and keypoints with low contrast or poorly localized on an edge are removed. Next, a consistent orientation is assigned to each keypoint and its magnitude is computed based on the local image gradient histogram, thereby achieving invariance to image rotation. At each keypoint descriptor, the contribution of local image gradients are sampled and weighted by a Gaussian, and then represented by orientation histograms. For example, the 16x16 sample image region and 4x4 array of histograms with 8 orientation bins are often used, thereby providing a 128-dimensional feature vector for each keypoint. Objects can be indexed and recognized using the histograms of keypoints in images. Numerous applications have been developed using the SIFT descriptors, including object retrieval [89] [90], and object category discovery [91]. Although the
SIFT approach is able to extract features that are insensitive to certain scale and illumination change, vision applications with large base line change entail the need of affine invariant point and region operators [92]. A performance evaluation among various local descriptors can be found in [93], and a study on affine region detectors is presented in [92]. Finally, SIFT-based methods are expected to perform better for objects with rich texture information as sufficient number of keypoints can be extracted. On the other hand, they also require sophisticated indexing and matching algorithms for effective object recognition [49] [94].

2.3. Object Tracking

Visual tracking is an important and active research area in computer vision. Accurate object segmentation and tracking under the constraint of low computational complexity presents a challenge. Our aim is to find solutions that are robust, simple, computationally feasible, modular, and easily adaptable to various applications.

Object tracking can be mainly classified in to three different types

1. Point Tracking
2. Kernel Tracking
3. Silhouette Tracking

2.3.1. Point Tracking

Objects detected in consecutive frames are represented by points, and the association of the points is based on the previous object state which
can include object position and motion. This approach requires an external mechanism to detect the objects in every frame. In point tracking multi-point correspondence is used. The object representation is simple and this type of tracking is used where the size of the object is small and simple.

Sethi and Jain [95] algorithm have consider two consecutive frames and is initialized by the nearest neighbor criterion. The correspondences are exchanged iteratively to minimize the cost. Veenman et al. [96] extend the work of Sethi and Jain [95], and Rangarajan and Shah [97] by introducing the common motion constraint for correspondence. The common motion constraint provides a strong constraint for coherent tracking of points that lie on the same object; however, it is not suitable for points lying on isolated objects moving in different directions. The algorithm is initialized by generating the initial tracks using a two-pass algorithm, and the cost function is minimized by Hungarian assignment algorithm in two consecutive frames. This approach can handle occlusion and misdetection errors, however, it is assumed that the number of objects are the same throughout the sequence, that is, no object entries or exits.

The Kalman filter has been extensively used in the vision community for tracking. Broida and Chellappa [98] used the Kalman filter to track points in noisy images. In stereo camera-based object tracking, Beymer and Konolige [110] use the Kalman filter for predicting the object’s
position and speed in $x-z$ dimensions. Rosales and Sclaroff [99] use the extended Kalman filter to estimate 3D trajectory of an object from 2D motion.

2.3.2. Kernel Tracking

Kernel refers to the object shape and appearance. For example, the kernel can be a rectangular template or an elliptical shape with an associated histogram. Objects are tracked by computing the motion of the kernel in consecutive frames.

Comaniciu and Meer [100] use a weighted histogram computed from a circular region to represent the object. They use the mean-shift procedure for tracking object. The mean-shift tracker maximizes the appearance similarity iteratively by comparing the histograms of the object, $Q$, and the window around the hypothesized object location, $P$. Histogram similarity is defined in terms of the Bhattacharya coefficient, $\sum_{u=1}^{b} P(u)Q(u)$, where $b$ is the number of bins. At each iteration, the mean-shift vector is computed such that the histogram similarity is increased. This process is repeated until convergence is achieved, which usually takes five to six iterations. For histogram generation, the authors use a weighting scheme defined by a spatial kernel which gives higher weights to the pixels closer to the object center.

Another approach to track a region defined by a primitive shape is to compute its translation by use of an optical flow method. Optical flow methods are used for generating dense flow fields by computing the flow
vector of each pixel under the brightness constancy constraint, \( I(x, y, t) - I(x + dx, y + dy, t + dt) = 0 \) [Horn and Schunk [101]]. This computation is always carried out in the neighborhood of the pixel either algebraically [Lucas and Kanade. [102]] or geometrically [Schunk [103]]. Extending optical flow methods to compute the translation of a rectangular region is trivial. In [104], Shi and Tomasi proposed the KLT tracker which iteratively computes the translation \((du, dv)\) of a region centered on an interest point.

2.3.3. Silhouette Tracking

Tracking is performed by estimating the object region in each frame. Silhouette tracking methods use the information encoded inside the object region. This information can be in the form of appearance density and shape models which are usually in the form of edge maps. Given the object models, silhouettes are tracked by either shape matching or contour evolution.

The representations chosen by the silhouette-based object trackers can be in the form of motion models (similar to point trackers), appearance models (similar to kernel trackers), or shape models or a combination of these. Object appearance is usually modeled by parametric or nonparametric density functions such as mixture of Gaussians or histograms. Object shape can be modeled in the form of contour subspace where a subspace is generated from a set of possible object contours obtained from different object poses [Blake and Isard...
Additionally, object shape can be implicitly modeled via a level set function where the grid positions are assigned at the distance generated from different level set functions corresponding to different object poses [Yilmaz et al. [106]]. Appearance-based shape representations are also commonly used by researchers who employ a brute force silhouette search. For edge-based shape representation, Hausdorff distance is the most widely used measure. However, Hausdorff measure is known for its sensitivity to noise. Hence, instead of using the maximum of distances, researchers have considered using an average of the distances [Baddeley [107]]. Occlusion handling is another important aspect of silhouette tracking methods. Usually methods do not address the occlusion problem explicitly. A common approach is to assume constant motion or constant acceleration where, during occlusion, the object silhouette from the previous frame is translated to its hypothetical new position. Few methods explicitly handle object occlusions by enforcing shape constraints [Mac-Cormick and Blake [108]; Yilmaz et al. [109]].

2.4. Pose Estimation

C.Yuan and H.Niemann [56] presented a neural network (NN) based system for recognition and pose estimation of 3D objects from a single 2D perspective view. He developed an appearance based neural approach for this task. First the object is represented in a feature vector derived by a principal component network. Then a NN classifier trained with
Resilient back propagation algorithm is applied to identify it. Next pose parameters are obtained by four NN estimation trained on the same feature vector. Under occlusion and noise, the average recognition rate is 77%.

In the method described by Dudani et al.[4], aircraft pose estimation takes place in two steps. In the first step, out of image plane object rotations are computed using Hu-moments and nearest neighbour search. In a second step, an analytical formula is used to compute in image plane object rotations from translation invariant moments. The feature vectors that are used in step one are composed of 7 hu-moments computed from object boundary pixels, and 7 hu-moments computed from object silhouette pixels. In order to make the nearest neighbour search more efficient, the dimensionality of the feature vectors is reduced from 14 to 5 by applying a Karhoene Loeve transform. In addition, a normalisation is carried out to obtain zero mean and unit variance for each of the vector components. The class recognition performance is approximately 95%. The size of the training set is minimised by taking into account the symmetry properties of an aircraft, and consists of approximately 500 images for each of the six aircraft types. The problem of pose ambiguity is not dealt with; it is simply assumed that this can be solved.

Wallace and Wintz[5] use a two step approach similar to the method described by Dudani, but instead of moments they use normalised
Fourier descriptors. A pose estimate is obtained by averaging the results from a k-nearest neighbour search. In this way, they achieve a pose estimation accuracy comparable to the results reported by Dudani, while using a reference feature vector library which is almost four times smaller. They emphasize the problems that can occur in designing a nonambiguous normalisation procedure for Fourier descriptors. In addition, they show that the effect of noise and image resolution changes on Fourier descriptor coefficients can be reduced by applying an appropriate filter.

Kui-yu Chang and Joydeep Ghosh[57] proposed a novel scheme using spherical manifolds for the simultaneous classification and pose estimation of 3-D objects from 2-D images. The spherical manifold imposes a local topological constraint on samples that are close to each other, while maintaining a global structure. Each node on the spherical manifold also corresponds nicely to a pose on a viewing sphere with 2 degrees of freedom. The proposed system is applied to aircraft classification and pose estimation.

Chen and Ho[6] describe a similar two step approach, but instead of silhouette moments they use Fourier descriptors of contours. They argue that, based on object symmetry, a complete set of all possible aircraft silhouettes can be obtained without generating all aircraft poses. This can be helpful in minimizing the reference database of feature vectors that is used by the pose estimation algorithm. Much attention is paid to
efficient nearest neighbour searching. They show that a method similar to nearest neighbour search can save computation time without sacrificing estimation accuracy. The study on aircraft identification and pose estimation carried out by Glais and Ayoun from Thomson-TRT-DEFENSE takes into account many problems that are encountered in realistic applications. The algorithm they describe is based on two different recognition approaches that are alternatively applied, depending on the quality and properties of the input images. The main part of the algorithm is based on the recognition approach described by Chen, but in case of input images in which target and background are not clearly separable, local image features and syntactic pattern recognition techniques are applied. In the first part of the algorithm, a watershed algorithm is used to separate object and background. Multiple separation hypotheses are generated, converted to Fourier descriptors, and compared to a library by means of nearest neighbour search. If no match is found during the search, it is assumed that the object separation was not successful. In this case, local image features and syntactic pattern matching are applied. Details of this approach are not provided in their paper. To further enhance the robustness of object recognition, evidence is accumulated over time. An estimate of the pose recognition reliability is computed on line. It is assumed that the pose ambiguity problem can be solved by using trajectory information. The performance of the system was investigated using simulations. During these simulations,
parameters such as background complexity, contrast, resolution, number of target types, and object pose were taken into consideration. In addition, they examined the sensitivity of the system performance to target shape modifications caused by presence or absence of ordinance systems. The overall conclusion is that good pose estimation and recognition performance can be achieved. No quantitative figures are provided.
3.1 Moving Object Segmentation Methods

In this chapter we study the performance of candidate technique for background subtraction and provide figure of effective. We try to find simple yet robust solution for real time implementation. The four different methods studied and implemented for background subtraction algorithms are:

- Frame differencing\[^{[20,21]}\].
- Approximated median filter\[^{[22,23]}\].
- Mixture of Gaussians\[^{[24]}\].
- Classification based on neural network.

3.1.1 Frame Differencing

Frame differencing is the simplest method for background subtraction\[^{[25]}\]. A background image without any moving objects of interest is taken as a reference image. Pixel value for each co-ordinate (x,y) for each color channel of the background image is subtracted from the corresponding pixel value of the input image. If the resulting value is greater than a particular threshold value, then that is a foreground pixel otherwise background.

For each frame I and for the reference image Iref, if for a particular pixel, \( |I_{val} - I_{ref_{val}}| > \text{Threshold} \) then that pixel is classified as foreground.
That is $I_{(i,j,k)} - I_{ref_{(i,j,k)}} > \text{Threshold}$ where $I_{(i,j,k)}$ is the co-ordinate $(i,j)$’s pixel value for $k^{th}$ color channel for the current image $I$ and $I_{ref_{(i,j,k)}}$ is for reference frame. An example of an input frame and the corresponding outputs are shown in Figures 3.1 through 3.4.

![Fig. 3.1 Input Frame](image)

As can be seen, a major flaw of this method is the selection of the threshold for objects with uniformly distributed intensity values (such as the side of a flying object). The interior pixels are interpreted as part of the background. Another problem is that objects must be continuously moving. If an object stays still for more than a frame period ($1/fps$), it becomes part of the background.
Fig. 3.2 Output Frame with threshold = 25

Fig. 3.3 Output Frame with threshold = 5 (low)
This method does have two major advantages. One obvious advantage is the modest computational load. Another is that the background model is highly adaptive. Since the background is based solely on the previous frame, it can adapt to changes in the background faster than any other method (at 1/fps to be precise).

3.1.2 Approximated Median Filter

In median filtering, the previous $N$ frames of video are buffered, and the background is calculated as the median of buffered frames. Then (as
with frame difference), the background is subtracted from the current frame and thresholded to determine the foreground pixels.

Median filtering has been shown to be very robust and to have performance comparable to higher complexity methods. However, storing and processing many frames of video (as is often required to track slower moving objects) requires an often prohibitively large amount of memory. This can be alleviated somewhat by storing and processing frames at a rate lower than the frame rate— thereby lowering storage and computation requirements at the expense of a slower adapting background.

McFarlane and Schofield propose a simple recursive filter to estimate the median. [23] In this scheme, the running estimate of the median is incremented by one if the input pixel is larger than the estimate, and decreased by one if smaller. This estimate eventually converges to a value for which half of the input pixels are larger than and half are smaller than this value, that is, the median. The input frame and the corresponding output are shown in Figure 3.5 and Figure 3.6 respectively.

The approximate median method works better at separating the entire object from the background. This is because the more slowly adapting background incorporates a longer history of the visual scene, achieving about the same result as if we had buffered and processed $N$ frames. But
this method trails behind the larger objects (the UAV). This is due to updating the background at a relatively high rate (30 fps).

Fig. 3.5 Input Frame

Fig. 3.6 Output Frame for threshold
3.2.3 Mixture of Gaussians (MoG)

The MoG method tracks multiple Gaussian distributions simultaneously. MoG has enjoyed tremendous popularity since it was first proposed for background modeling in [24]. Similar to the non-parametric model MoG maintains a density function for each pixel. Thus, it is capable of handling multi-modal background distributions. On the other hand, since MoG is parametric, the model parameters can be adaptively updated without keeping a large buffer of video frames. Our description of MoG is based on the scheme described in [19]. The pixel distribution \( f(I_t = u) \) is modeled as a mixture of \( K \) Gaussians:

\[
f(I_t = u) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(u; \mu_{i,t}, \sigma_{i,t})
\]

[3.1]

where \( \eta(u; \mu_{i,t}, \sigma_{i,t}) \) is the \( i \)-th Gaussian component with intensity mean \( \mu_{i,t} \) and standard deviation \( \sigma_{i,t} \). \( \omega_{i,t} \) is the portion of the data accounted for by the \( i \)-th component. Typically, \( K \) ranges from three to five, depending on the available storage. For each input pixel \( I_t \), the first step is to identify the component \( \hat{i} \) whose mean is closest to \( I_t \). Component \( \hat{i} \) is declared as the matched component if

\[
|I_t - \mu_{\hat{i},t-1}| \leq D \sigma_{\hat{i},t-1}
\]

where \( D \) defines a small positive deviation threshold. The parameters of the matched component are then updated as follows:
\[ \omega_{i,t} = (1 - \alpha)\omega_{i,t-1} + \alpha \]
\[ \mu_{i,t} = (1 - \rho)\mu_{i,t-1} + \rho I_t \]
\[ \sigma_{i,t}^2 = (1 - \rho)\sigma_{i,t-1}^2 + \rho(I_t - \mu_{i,t})^2, \]

where \( \alpha \) is a user-defined learning rate with \( 0 \leq \alpha \leq 1 \). \( \rho \) is the learning rate for the parameters and can be approximated as follows:

\[ \rho \approx \frac{\alpha}{\omega_{i,t}} \]

If no matched component can be found, the component with the least weight is replaced by a new component with mean \( I_t \), a large initial \( \sigma_0 \) variance and a small weight \( \omega_0 \). The rest of the components maintain the same means and variances, but lower their weights to achieve exponential decay:

\[ \omega_{i,t} = (1 - \alpha)\omega_{i,t-1} \]

Finally, all the weights are renormalized to sum up to one. To determine whether \( I_t \) is a foreground pixel, we first rank all components by their values of \( \omega_{i,t} / \sigma_{i,t} \). Higher-rank components thus have low variances and high probabilities, which are typical characteristics of
background. If \( i_1; i_2; \ldots; i_K \) is the component order after sorting, the first \( M \) components that satisfy the following criterion are declared to be the background components:

\[
\sum_{k=s_1}^{i_M} \omega_{k,t} \geq \Gamma,
\]

where \( \Gamma \) is the weight threshold. It is declared as a foreground pixel if it is within \( D \) times the standard deviation from the mean of any one of the background components. Note that the above formulation can be easily extended to handle color data. The computational complexity and storage requirement of MoG is linear in terms of the number of components \( K \). Recent development in MoG technologies include a sensitivity analysis of parameters[26], improvements in complexity and adaptation[27,28,29], and an extension to construct a panoramic background[30]. The input frame and the corresponding output are shown in Figure 3.7 and Figure 3.8 respectively.

3.1.4 Feature Calculation and Classification Based on Neural Network

The basic idea of the method is to calculate the texture features of the images such as Haar wavelet energy, second and third moments. Feed the features to the neural network and train them. The database consists of different 50 sky images with or without clouds and 250 parts of aircraft images.
The original image is obtained from the video input and decomposed into subimages images to calculate the texture features of the
subimages such as Haar wavelet energy, second and third moments. Feed the texture features to the neural network. Based on the output of the neural network the subimage is removed if it is cloud texture.

Haar Wavelet Transform

In mathematics, the Haar wavelet is a certain set of functions. This sequence was proposed in 1909 by Alfréd Haar[54]. Haar used these functions to give an example of a countable orthonormal system for the space of square-integrable functions on the real line. Fig. 3.9 shows the haar mother wavelet.

![Fig.3.9. The Haar wavelet](image)

The Haar wavelet's mother wavelet function $\psi(t)$ can be described as

$$
\psi(t) = \begin{cases} 
1 & 0 \leq t < 1/2, \\
-1 & 1/2 \leq t < 1, \\
0 & \text{otherwise.}
\end{cases}
$$

[3.6]
and its scaling function $\phi(t)$ can be described as

$$
\phi(t) = \begin{cases} 
1 & 0 \leq t < 1, \\
0 & \text{otherwise.}
\end{cases}
$$

Haar Wavelet properties

1. Any continuous real function can be approximated by linear combinations of $\phi(t), \phi(2t), \phi(4t), \ldots, \phi(2^k t), \ldots$ and their shifted functions. This extends to those function spaces where any function therein can be approximated by continuous functions.

2. Any continuous real function can be approximated by linear combinations of the constant function $\psi(t), \psi(2t), \psi(4t), \ldots, \psi(2^k t), \ldots$, and their shifted functions.

3. Orthogonality in the form

$$
\int_{-\infty}^{\infty} 2^m \psi(2^m t - n) \psi(2^m t - n_1) \, dt = \delta_{m,m_1} \delta_{n,n_1}.
$$

Here $\delta_{i,j}$ represents the Kronecker delta. The dual function of $\psi(t)$ is $\psi(t)$ itself.

4. Wavelet/scaling functions with different scale $m$ have a functional relationship:

$$
\phi(t) = \phi(2t) + \phi(2t - 1)
$$
\[ \psi(t) = \phi(2t) - \phi(2t - 1) \]  \hfill [3.9]

5. Coefficients of scale \( m \) can be calculated by coefficients of scale \( m+1 \):

If

\[ \chi_w(n, m) = 2^{m/2} \int_{-\infty}^{\infty} x(t) \phi(2^m t - n) \, dt \]  \hfill [3.10]

and

\[ X_w(n, m) = 2^{m/2} \int_{-\infty}^{\infty} x(t) \psi(2^m t - n) \, dt \]  \hfill [3.11]

Then

\[ \chi_w(n, m) = \sqrt{\frac{1}{2}} (\chi_w(2n, m + 1) + \chi_w(2n + 1, m + 1)) \]  \hfill [3.12]

The Haar transform is the simplest of the wavelet transforms. This transform cross-multiplies a function against the Haar wavelet with various shifts and stretches, like the Fourier transform cross-multiplies a function against a sine wave with two phases and many stretches.

\[
H_4 = \frac{1}{\sqrt{4}} \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
\sqrt{2} & -\sqrt{2} & 0 & 0 \\
0 & 0 & \sqrt{2} & -\sqrt{2}
\end{bmatrix}
\]  \hfill [3.13]

The Haar transform can be thought of as a sampling process in which rows of the transform matrix act as samples of finer and finer resolution.

**Why Haar wavelet transform**

The Haar wavelet transform has a number of advantages [31]:

1. It is conceptually simple.
2. It is fast.

3. It is memory efficient, since it can be calculated in place without a temporary Array.

4. It is exactly reversible without the edge effects that are a problem with other Wavelet transforms.

Based on 1) – 3) we select haar for testing.

Texture Features

In this work two sets of DWT derived features are considered.

1. Energies of wavelet coefficients calculated in subbands at successive scales. According to Eq. (3.14)

\[ E_{\text{subband, scale}} = \sum_{x,y \in \text{ROI}} \left( d_{x,y}^{\text{subband}} \right)^2 \]

\[ n \]

[3.14]

where \( n \) is the number of pixels in ROI, both at given scale and subband and \( d_{x,y} \) are wavelet coefficients of the wavelet decomposition at given scale and subband.

2. Ratio of variance and mean of the wavelet coefficients i.e.

\[ \frac{V_{\text{subband, scale}}}{M_{\text{subband, scale}}} \]

[3.15]

Where, \( M_{\text{subband, scale}} = \frac{1}{n} \sum_{i=1}^{n} d_{i}^{\text{subband}} \)

\[ V_{\text{subband, scale}} = \frac{1}{n} \sum_{i=1}^{n} \left( d_{i}^{\text{subband}} - M_{\text{subband, scale}} \right)^2 \]
To compute the wavelet features in the first step the algorithm with Haar wavelet functions is applied to the whole image. As a result 3 detail subbands are generated at each scale. (Fig.3.10.)

In the next step, energy (3.6) of $d^{LH}$, $d^{HL}$ and $d^{HH}$ is calculated at any considered scale in marked ROIs. Where the region of interest (ROI) is a common property that is supported by all of the standard operators. The ROI is simply a description of some portion of an image.

ROIs are reduced in successive scales in order to correspond to subband image dimensions. In a given scale the energy is calculated only if ROI at this scale contains at least 4 points. Output of this procedure is a vector of features containing energies of wavelet coefficients calculated in subbands at successive scales. In this we consider upto 1 level of scaling.

**Fig.3.10.** Subband images.
The two-dimensional moment invariants are invariant with respect to translation, scale and rotation of the shape. The moments which have the property of invariant image recognition as well as image reconstruction given the moment descriptors have been introduced by Hu[32].

The normalised second and third central moments can be calculated and are defined by:

\[
\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{21}^2
\]

\[
\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2
\]

(3.16)

Classification using wavelet and moments features and Neural Networks

Feature selection

Selection of features plays a key role in classifying aircraft texture. For classification of objects in the segmented images we look for features such as Haar wavelet energy, mean, variance of wavelet coefficients at level 1 and second and third moments are considered in this context. Once the features are computed for every object, they are normalized between zero and one in order to make them compatible with other features before feeding to neural networks.

Neural Networks design

For designing a neural network, several factors have to be taken care to do proper classification for example number of nodes in each layer, number of hidden layers and the type of training function. The choice of
number of hidden layers and nodes in the hidden layers depends on network application. However the number of hidden layers is chosen based on training of the network using various configurations of layers. The configuration with a fewest number of layers and nodes yielding minimum RMS error quickly and efficiently is preferred. The number of neurons in the input and output layers depends on the application. The number of neurons in the hidden layers can be determined by experimentation. Too few neurons prevent it from correct mapping inputs to outputs and too many neurons impede generalization and increase training time but they allow the network to memorize patterns without extracting pertinent features for generalization.

The classifier used in this research is a three-input three-layer feed forward neural network with a single output node. Fig.3.11. shows the architecture of the NN used in our design. The neurons are saturated linear functions for input layer and logarithmic sigmoid functions for hidden and output layers.

During the training stage, the features computed for every texture are fed to the network in the form of an input matrix. The desired output corresponding to the respective feature set is also entered as a target vector.

Once the weights and biases of neurons associated with network are initialized, the network is trained to produce output “1” indicating presence of aircraft texture and “0” indicating cloud texture case. During
training, the weights and biases are altered iteratively to minimize the network performance function that is MSE.

In view of the fact that machine learning methods specifically NN utilize large training set and are highly parameterized, a very small neural network comprising of only three significant features is considered. Due to very small feature set, there is no need of using any additional method for selection of features. The training set comprises normalized features taken from 250 cases including 200 aircraft texture and 50 cloud textures.

NN based Classification

Once training is completed, the testing of applied neural network is carried out. The fig.3.12. below shows the block diagram for testing.
Consider an image \( I \) of size \( m \times n \) as shown Fig.3.13. below. The basic method is to decompose the original image in subimages images of size \( k \times k \) and calculate the texture features of the subimages such as Haar wavelet energy, second and third moments. Feed the texture features to the neural network. Based on the output of the neural network the subimage is removed if it is cloud texture. In the experiment we considered subimages instead of 4x4 and 8x8 for comparison.

![Fig.3.12. Background segmentation(subtraction)](image)

In object classification problem, the four quantities of results category are given below.

(i) True Positive \((TP)\)  
Classify a flying object image into class of flying object.

(ii) True Negative \((TN)\)  
Misclassify a flying object image into class of Non-flying object.
(iii) False Positive \((FP)\)
Classify a non-flying object image into class of non-flying object.

(iv) False Negative \((FN)\)
Misclassify a non-flying object image into class of flying object.

The objective of any classification is to maximize the number of correct classification denoted by True Positive Rate \((TPR)\) and False Positive Rate \((FPR)\) where by minimizing the wrong classification denoted by True Negative Rate \((TNR)\) and False Negative Rate \((FNR)\).
\[
    TPR = \frac{\text{Number of true positive (TP)}}{\text{Total number of positive in data set (nP)}}
\]

\[
    TNR = \frac{\text{Number of true negative (TN)}}{\text{Total number of negative in data set (nN)}}
\]

\[
    FPR = \frac{\text{Number of false positive (FP)}}{\text{Total number of positive in data set (nP)}}
\]

\[
    FNR = \frac{\text{Number of false negative (FN)}}{\text{Total number of negative in data set (nF)}}
\]

Overall Classification Accuracy (4X4) = \(\frac{TPR + FPR}{2}\) = 90.5 %

Overall Classification Accuracy for (8X8) = \(\frac{TPR + FPR}{2}\) = 85.3 %

<table>
<thead>
<tr>
<th>subImage</th>
<th>Classifying Positive Images</th>
<th>Classifying Negative Images</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Flying object</td>
<td>Non-flying object (Clouds and sky)</td>
</tr>
<tr>
<td>TPR</td>
<td>TNR</td>
<td>FPR</td>
</tr>
<tr>
<td>4x4</td>
<td>88.6%</td>
<td>11.4%</td>
</tr>
<tr>
<td>8x8</td>
<td>84.2%</td>
<td>15.8%</td>
</tr>
</tbody>
</table>

Table 3.1 Background subtraction Classification results.
From the results it can be observed that TPR, TNR, FPR has 2 to 4% difference but the FNR has be almost doubled. For tracking purpose we can consider 8x8 subimage size but for pose estimation 4x4 subimage gives better results.

Fig.3.14.a. Original Image

Fig.3.14. shows the an example of the background subtraction using the proposed method.
Fig. 3.14.b. 4X4 Subimage output

Fig. 3.14.c. 8X8 Subimage output
Conclusion

The frame differencing results in a very noisy result due to its dependency on the threshold parameter. The second method, with an approximate median, yields significantly increased accuracy at not much more computation. It had a little trouble with quickly changing light levels, but handled them better than mixture of Gaussians. The third method Mixture of Gaussians, gives us good performance, but presents a tricky parameter optimization problem. And wavelet and neural network method, the most complex of the methods, gives us good performance compared to all but it takes more time to perform background subtraction compared to all other methods. Therefore, the approximate median filter is seen as a good choice for the application compared to all methods.
4.1. Mean Shift Tracking

One possible approach is to use the mean shift algorithm to localize the target object. The mean shift algorithm is a robust statistical method which finds local maxima in the probability distribution. It works with a search window that is positioned over a section of the distribution. Within this search window the maximum can be determined by a simple average computation. Then the search window is moved to the position of this maximum and the average computation is repeated again. This procedure is repeated until the algorithm finds a local maximum and converges.

To apply the mean shift algorithm for object tracking it is necessary to represent the data of video frames as a probability distribution. Every pixel in a frame gets a probability value $P(u,v)$, depending on its color/intensity, which indicates how likely it is that the related pixel belongs to the target object. Using this probability values a frame can be represented as a 2D probability distribution and the mean shift algorithm can be applied. Mean shift is used in color-based object tracking because it is simple and robust. The best results can be achieved if the following conditions are fulfilled:

- The target object is mainly composed of one color.
• The target object does not change its color.
• Illumination does not change dramatically.
• There are no other objects in the scene similar to the target object.
• The color of the background differs from the target object.
• There is no full occlusion of the target object.

There are numerous approaches employing the mean shift algorithm in object tracking. Bradski presents in [33] his so-called CAMShift (Continuously Adaptive Mean Shift) method, Allan et al. improve in [34] the CAMShift method by using a better target model, and Cominiciu et al. [35] propose to use a candidate model in addition to the target model. The mean shift algorithm is often combined with other methods to improve the tracking results. Han et al. [36] use the mean shift algorithm in combination with a double model filter to obtain robust results in scenes with abrupt and fast motion. In [37], Jeong et al. propose to use a Gaussian-cylindroids color model to increase robustness against illumination changes. Wang and Yagi [38] describe an approach using not only color but also shape features with mean shift. Besides object tracking, the mean shift algorithm can also be used for smoothing and segmentation [39]. In [40], Zivkovic et al. proposed a 5-DOF color-histogram-based tracking method that estimates the position of the tracked object but also simultaneously estimates the ellipse that approximates the shape of the object.
4.1.1 Implementation of the Mean-Shift Tracking (MST)

This section describes three possible variants of the mean shift object tracking method. They were chosen because they differ only by their target model and use no additional features besides the color of the target object. As these three approaches are similar, it is interesting to see how their results differ in the experiments.

4.1.2 Mean Shift Tracking- 1 (MST-1)

Zivkovic proposed a 5-DOF color-histogram-based tracking method that estimates the position of the tracked object but also simultaneously estimates the ellipse that approximates the shape of the object. The new algorithm solves the mentioned problem of adapting the ellipse in an efficient way.

We assume that the shape of a non-rigid object is approximated by an ellipsoidal region in an image. Initially the object is selected manually or detected using some other algorithm, background subtraction for example. Let \( \bar{x}_i \) denote a pixel location and \( \bar{\theta}_0 \) the initial location of the center of the object in the image. The second order moment can be used to approximate the shape of the object:

\[
V_0 = \sum_{\text{object pixels}} (\bar{x}_i - \bar{\theta}_0) (\bar{x}_i - \bar{\theta}_0)^T
\]  

[4.1]
Further, the color histogram is used to model the object appearance. Let the histogram have M bins and let the function $b(\bar{x}_i): \mathbb{R}^2 \rightarrow 1,...,M$ be the function that assigns a color value of the pixel at location $\bar{x}_i$ to its bin. The color histogram model of the object consists then of the M values of the M bins of the histogram $\bar{\sigma} = [o_1,...,o_M]^T$. The value of the m-th bin is calculated by:

$$O_m = \sum_{i=1}^{NV_0} N(\bar{x}_i; \bar{\theta}_0, V_0) \delta[b(\bar{x}_i) - m]$$  \hspace{1cm} [4.2]

where $\delta$ is the Kronecker delta function. We use the Gaussian kernel N to rely more on the pixels in the middle of the object and to assign smaller weights to the less reliable pixels at the borders of the objects. We use only the NV0 pixels from a finite neighborhood of the kernel and the pixels further than 2:5-sigma are disregarded.

Input: the object model $\bar{\sigma}$, its initial ($k = 0$) location $\bar{\theta}^{(k)}$ and shape defined by $V(k)$.

1. Compute the values of the color histogram of the current region defined by $\theta(k)$ and $V(k)$ from the current frame using equation.

$$r_m(\bar{\theta}, V) = \sum_{i=1}^{NV} N(\bar{x}_i; \bar{\theta}, V) \delta[b(\bar{x}_i) - m]$$  \hspace{1cm} [4.3]
2. Calculate weights using equation

\[
    w_i = \sum_{m=1}^{M} \sqrt{\frac{o_m}{r_m(\theta^{(k)}, \nu^{(k)})}} \delta[b[\tilde{x}_i - m] \quad [4.4]
\]

3. Calculate \( q_i \)-s using equation

\[
    q_i = \frac{w_i N(\tilde{x}_i; \theta^{(k)}, \nu^{(k)})}{\sum_{i=1}^{N} w_i N(\tilde{x}_i; \theta^{(k)}, \nu^{(k)})} \quad [4.5]
\]

where \( q_i \)-s are arbitrary constants that meet the following requirements:

\[
    \sum_{i=1}^{N} q_i = 1 \text{ and } q_i \geq 0 \quad [4.6]
\]

4. Calculate new position estimate \( \theta^{(k+1)} \) using equation

\[
    \tilde{\theta}^{(k+1)} = \sum_{i=1}^{N} q_i \tilde{x}_i = \frac{\sum_{i=1}^{N} w_i N(\tilde{x}_i; \theta^{(k)}, \nu^{(k)})}{\sum_{i=1}^{N} w_i N(\tilde{x}_i; \theta^{(k)}, \nu^{(k)})} \quad [4.7]
\]

5. Calculate new variance estimate \( V^{(k+1)} \) using equation

\[
    \tilde{V}^{k+1} = \beta \sum_{i=1}^{N} q_i (\tilde{x}_i - \tilde{\theta}^{(k)}) (\tilde{x}_i - \tilde{\theta}^{(k)})^T \quad [4.8]
\]

6. If no new pixels are included using the new elliptical region defined by the new estimates \( \theta^{(k+1)} \) and \( V^{(k+1)} \) stop, otherwise set \( k \leftarrow k + 1 \) and
The procedure is repeated for each frame. In the simplest version the position and shape of the ellipsoidal region from the previous frame are used as the initial values for the new frame.

In Fig. 4.1, a few frames are shown from the sequence of 1010 frames, wherein the position of an aircraft is changing rapidly. In the above figure we can see that the algorithm can track the object and also adapt to the shape and scale change of the object.
4.1.3 Mean Shift Tracking-2 (MST-2)

A rectangular window is defined about the region of interest in an initial frame. Then the mean shift algorithm is applied to separate the tracked object from the background in LUV color space. As the object moves, an unusual kernel weighted by the Chamfer distance transform improves the accuracy of target representation and localization, minimizing the distance between two color distributions using the Bhattacharyya coefficient. In tracking an object through a color image sequence, we assume that we can represent it by a discrete distribution of samples from a region in color space, localized by a kernel whose centre defines the current position. Hence, we want to find the maximum
in the distribution of a function $\rho$, that measures the similarity between the weighted color distributions as a function of position (shift) in the candidate image with respect to a previous model image. If we have two sets of parameters for the respective densities $p(x)$ and $q(x)$, the Bhattacharyya coefficient [4.9] is an approximate measurement of the amount of overlap, defined by:

$$\rho = \int \sqrt{p(x)q(x)} \, dx$$  \hspace{1cm} [4.9]

Since we are dealing with discretely sampled data from color images, we use discrete densities stored as $m$-bin histograms in both the model and candidate image. The discrete density of the model is defined as:

$$q = \{q_u\}, u = 1, 2, \ldots, m \quad \sum_{u=1}^{m} q_u = 1$$  \hspace{1cm} [4.10]

Similarly, the estimated histogram of a candidate at a given location $y$ in a subsequent frame is:

$$p(y) = \{p_u(y)\}, u = 1, 2, \ldots, m \quad \sum_{u=1}^{m} p_u = 1$$  \hspace{1cm} [4.11]

According to the definition the sample estimate of the Bhattacharyya coefficient is given by:
Algorithm

Given: the target model \( \{ \hat{q}_u \}_{u=1}^m \) and its location \( \hat{y}_0 \) in the previous frame.

1. Initialize the location of the target in the current frame with \( \hat{y}_0 \), compute \( \{ \hat{p}_u (\hat{y}_0) \}_{u=1}^m \) and evaluate

\[
\rho(\hat{y}) = \rho(p(\hat{y}), q) = \sum_{u=1}^m \sqrt{p_u(\hat{y}) q_u} \quad [4.12]
\]

\[
\rho[\hat{p}(\hat{y}_0), q] = \sum_{u=1}^m \sqrt{\hat{p}_u(\hat{y}_0) \hat{q}_u} \quad [4.13]
\]

2. Derive the weights \( \{ w_i \}_{i=1}^{n_h} \) according to equation

\[
w_i = \frac{1}{\sqrt{\rho_u(\hat{y}_0)}} \delta [b(X_i) - u]. \quad [4.14]
\]

3. Find the next location of the target candidate according to equation

\[
\hat{y}_1 = \frac{\sum_{i=1}^{n_h} X_i w_i g(\frac{\hat{y}_0 - X_i}{h})^2}{\sum_{i=1}^{n_h} w_i X_i g(\frac{\hat{y}_0 - X_i}{h})^2} \quad [4.15]
\]

4. Compute \( \{ \hat{p}_u (\hat{y}_1) \}_{u=1}^m \), and evaluate
\[
\rho[\hat{\mathbf{y}}_1, \hat{q}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u(\hat{y}_1)\hat{q}_u}
\]  \hspace{1cm} [4.16]

5. While \( \rho[\hat{\mathbf{y}}_1, \hat{q}] < \rho[\hat{\mathbf{y}}_0, \hat{q}] \)

\[
\text{Do } \hat{\mathbf{y}}_1 \leftarrow \frac{1}{2} (\hat{\mathbf{y}}_0 + \hat{\mathbf{y}}_1)
\]

Evaluate \( \rho[\hat{\mathbf{y}}_1, \hat{q}] \)

6. If \( \|\hat{\mathbf{y}}_1 - \hat{\mathbf{y}}_0\| < \epsilon \) Stop.

Otherwise set \( \hat{\mathbf{y}}_0 \leftarrow \hat{\mathbf{y}}_1 \) and go to Step 2.

In Fig. 4.2, the tracked objects is shown in the search window. While the accuracy is as for MST-1, one can see the amount of background incorporated into the search window. The window size is constant, so it can't adapt to the scale changes.
4.2 Scale Invariant Feature Transform [SIFT] Based Tracking

The general methodology is to detect SIFT features [48] for objects and then repeatedly update the SIFT descriptors throughout the course of the video sequence. First, the user gives the template to track. The system then detects and saves all the SIFT features within that user-selected region. Then the system begins processing all of the successive frames in the video sequence. In each new frame it looks within a padded region around the objects location in the previous frame to locate the object in the new frame. The key here is that the padding parameter must be set large enough that the object will never move faster than the padding window size in two successive frames. The padding parameter is a free parameter that must be set by the user. For all of my experiments we set the padding parameter to be between 9 and 12 pixels.
Since objects in videos are likely to change appearance during the course of the video (either due to motion, rotation, or lighting variations) we decided to include a re-calibration stage at repeated intervals throughout the video. So that after every N frames, the system redefines what SIFT features describe the object it is tracking. The update-rate parameter must be chosen by the user. For all of the sequences we tested in this project, the update-rate was set to recalibrate every 30 frames.

Also, we added an emergency situation detector which detects that no features were found in the current frame and it tells the system to do an emergency recalibration as soon as a descriptor is noticed in any subsequent frames. This emergency updater becomes necessary when an object undergoes a very rapid change in appearance.

4.2.1 Algorithm Description

- Template image of object in first frame
- Detect SIFT features for each object
- Set recalibration counter to 30 for each object
- loop through each frame in the sequence
  - loop through each object we are tracking
    - Detect and match SIFT features for current object looking only within padded region around where it was last detected
      - if (no matches were found) then
        - Set recalibration counter to zero for current object
• Expand the previous location of the object by the padding size
  else

• Update current object’s location by drawing a box around all the
detected features in the current frame.

• Decrease recalibration counter for current object by one

• if (recalibration counter < 1) then

  Re-analyze the SIFT features for the current object

In Fig. 4.3, a few frames are shown from the sequence of 1010 frames, wherein the position of an aircraft is changing rapidly. In the figure we can see that the algorithm can track the object and also adapt to the shape and scale change of the object.
4.3. Harris Based Tracking

The algorithm for tracking is same as SIFT but the here the tracking is based on harris cornet features.

4.3.1. Harris Corner Detector[58]

Corners are the intersections of two edges of sufficiently different orientations. They are important two dimensional image features to represent object shapes. Corners are stable across image sequences and useful in image matching for stereo and object tracking for motion, therefore playing an important role in matching, pattern recognition, robotics, and measurement.

Corners are generally located in the region with large intensity variations in every direction. The instrument to detect corners lies in
image derivatives. Let $I_x$ and $I_y$ be image gradients in horizontal and vertical directions, we can define a matrix $C$ as

$$C = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}$$

[4.17]

where the sum are taken over a neighborhood of the pixel in consideration. This matrix characterizes the structure of the image gray level patterns. In fact, the geometric interpretation of the gray levels is encoded in the eigenvector and eigenvalues of the matrix. $C$ is symmetric and it has two nonnegative eigenvalues. The eigenvectors of $C$ encode the directions, while the eigenvalues encode the variational strength. A corner is detected if the minimum of the two eigenvalues is larger than a threshold. An alternative for detecting corners is: if

$$\det(C) - k \cdot \text{trace}(C^2)$$

[4.18]

is larger than a threshold, where $k$ is a small number (0.04). This method is called *Harris Corner Detector*, which is good for detecting corners with orthogonal edges.

4.3.2. Point Matching
Correlation-based method is utilized to establish the correspondence between the first and second image points. The principle of this method is to find two corresponding points based on the intensity distributions of their neighborhoods. The similarity between two neighborhoods is measured by their cross-correlation. The underlying assumptions include:

- Corresponding image regions are similar
- Point and distant or single light source
- Corresponding points are visible from both viewpoints

Given an image point on the left image, the task is to locate another point in a specified region on the second image that is maximally correlated with the one on the first. For each first image pixel, its correlation with a second image pixel is determined by using a small correlation window of fixed size.

\[
c(d) = \sum_{k=-W}^{W} \sum_{l=-W}^{W} \Psi(I_i(i+k, j+l), I_r(i+k-d_1, j+l-d_2))
\]

[4.19]

where \((i, j)\) are the coordinates of the first image pixel.

\(\vec{d} = (d_1, d_2)^T\) is the relative displacement between the first and second image pixels.

\(2W+1\) is the width of the correlation window.
$I_l$ and $I_r$ are the intensities of the first and second image pixels respectively.

$\Psi(u, v) = -(u - v)^2$ is the SSD correlation function.

By moving the correlation window in certain region of the right image, the corresponding point is set as the pixel that has the highest correlation value.

4.3.3. Computing Fundamental Matrix F and Essential Matrix E

The fundamental matrix was found through the 8 point algorithm.

1. The constraint matrix $A$ was created from the 2 lists of matched corners [41][42].

\[
A = p_l^T F p_r = 0 \quad [4.20]
\]

2. SVD was used to solve the linear system

\[
SVD(A) = UDV^T \quad [4.21]
\]

3. The fundamental matrix is then formed from the column of $V$ corresponding to the smallest singular value of $S$.

4. To make sure that the fundamental matrix was of rank 2, SVD is repeated on the fundamental matrix itself and the new fundamental matrix is formed from the 2 largest singular values of the new $S$. 
4.3.4. RANSAC (Random Sample Consensus) – Used to remove outliers and recomputed the fundamental matrix based on the remaining inliers [41][43][44].

Assume:

1. The parameters can be estimated from N data points.
2. There are a total of M data points.
3. The probability of a random data point being part of a good model is Pg.
4. The probability that no good fits will be found is Pf.

Then, the algorithm:

1. Selects N random data points.
2. Model the N data points with a fundamental matrix.
3. Find how many data points of M fit the model with parameters from #2.
4. If the number of good data points is above a threshold K, exit with success.
5. Repeat #1 through #4 L times.
6. Otherwise return with fail.
In Fig. 4.4, a few frames are shown from the sequence of 1010 frames, wherein the position of an aircraft is changing rapidly. In the above figure we can see that the algorithm can track the object and also adapt to the shape and scale change of the object.
4.4. Fast Full Search Based on FFT Algorithms

The full search technique was originally described by Jain and Jain [45]. Each image frame is divided into a fixed number of usually square blocks $B \times B$. For each block $g$ in the current frame $I_c$, a search is made in the reference frame $I_r$ over a search area $f$ within a fixed-sized of search window $\pm w$ (see Figure 4.5).

![Figure 4.5. Search area in full search block matching algorithm](image)

The search is for the best matching block, to give the least prediction error, usually minimizing either sum absolute difference (SAD), or sum square difference (SSD). This latter can be expressed as follows:
where \((dx, dy)\) is the motion vector candidate. For simplicity of notations, we assume that \(dx\) and \(dy\) are in \([0, 2w]\). Note that \(dx\) between 0 and \(w\) indicates a negative displacement, meanwhile \(dx\) between \(w\) and \(2w\) indicates a positive one and the same for \(dy\).

Let:

\[
T_1(dx, dy) = \sum_{l=0}^{B-1} \sum_{k=0}^{B-1} f^2(k + dx, l + dy),
\]

\[4.23\]

\[
T_2(dx, dy) = \sum_{l=0}^{B-1} \sum_{k=0}^{B-1} -2f(k + dx, l + dy)g(k, l),
\]

\[4.24\]

\[
T_3 = \sum_{l=0}^{B-1} \sum_{k=0}^{B-1} g^2(k, l),
\]

\[4.25\]

Then the SSD metric can be written as:

\[
SSD(dx, dy) = T_1(dx, dy) + T_2(dx, dy) + T_3.
\]

\[4.26\]
then

\[ T_1(d_x, d_y) = S(d_x + m d_y + n). \] \hfill [4.27]

Thus the SSD metric can be expressed as below:

\[ \text{SSD}(d_x, d_y) = S(d_x + m d_y + n) + T_2(d_x, d_y) + T_3. \] \hfill [4.28]

The last term \( T_3 \) is independent of the motion vector \((dx, dy)\).
Therefore, minimizing SSD metric corresponds to minimize:

\[ S(d_x + m d_y + n) + T_2(d_x, d_y). \] \hfill [4.29]

4.4.1 Algorithm Description

(1) Computing the sum square blocks for the whole image.

(2) Selecting first and second input data blocks \( g_1 \) and \( g_2 \) in said current image, and selecting first and second input search areas \( f_1 \) and \( f_2 \) in said reference image.

(3) Converting the input data blocks to a complex data block as below:

\[ g_c(x, y) = g_1(x, y) + j g_2(x, y), \] \hfill [4.30]
where $j$ is the square root of $-1$.

Note here that $g1$ and $g2$ have been padded by zeros to the size $N^2$. For simplicity of notation, $N^2$ denotes the size of the search areas.

(4) Let $Gc$ be the FFT of $gc$, determining a first frequency domain data block $G1$ and a second frequency domain data block $G2$ from $Gc$ as follows:

$$G_1(u,v) = \frac{G_c(u,v) + G_c^*(N-u,N-v)}{2},$$  

[4.31]

And

$$G_2(u,v) = \frac{G_c(u,v) - G_c^*(N-u,N-v)}{2j}.$$  

[4.32]

These formulas can be obtained easily by utilizing the symmetrical properties of the FFT process by which real inputs produce even real and odd imaginary outputs and imaginary inputs produce odd real and even imaginary outputs;

(5) In the same way as we have computed $G1$ and $G2$, determining the FFT of the search areas $F1$ and $F2$.

(6) Computing the following surface:
(7) Inverse transforming the complex resultant surface to a resultant spatial blocks having real and imaginary pads;

(8) Determining first and second cross-correlations between said input data blocks and said input search areas by separating the real and imaginary pads of said resultant spatial block wherein said real part is the first cross correlation between the first input data and search area and said imaginary part is the second cross correlation between the second input data and the corresponding search area.

(9) Repeating steps (2) through (8) for other of said plurality of data blocks so as to generate a plurality of motion vectors.

In Fig.4.6, a few frames are shown, from the sequence of 1010 frames, where the position of an UAV is changing rapidly. In the Fig. we can see that the algorithm can track the object correctly.
Fig. 4.6. Tracking through the sequence using SSD FFT

Conclusion

The tracker was implemented in two versions based on tradeoff between accuracy and speed. The results show that SIFT- features are really scale invariant, more robust to illumination changes compared to other
algorithms and give a good estimation of the aircraft and is quite stable to occlusions. But it is slow compared to mean shift and SSD FFT algorithms.
OBJECT DETECTION AND POSE IDENTIFICATION

In chapter 2, a survey is presented, which category this method belongs to and why we select this method.

5.1 Moment Functions for Object Identification.

5.1.1 Moment Based Method

Moment Invariants have been frequently used as features for image processing, remote sensing, shape recognition and classification. Moments can provide characteristics of an object that uniquely represent its shape. Invariant shape recognition is performed by classification in the multidimensional moment invariant feature space. Several techniques have been developed that derive invariant features from moments for object recognition and representation. These techniques are distinguished by their moment definition, such as the type of data exploited and the method for deriving invariant values from the image moments.

Two-dimensional moment invariants are invariant with respect to translation, scale and rotation of the shape. The moments which have the property of invariant image recognition as well as image reconstruction given the moment descriptors have been introduced by Hu [32].
Translation invariance is achieved by computing moments that are normalized with respect to the centre of gravity so that the centre of mass of the distribution is at the origin (central moments). Size invariant moments are derived from algebraic invariants but these can be shown to be the result of simple size normalization. From the second and third order values of the normalized central moments a set of seven invariant moments can be computed which are independent of rotation.

Traditionally, moment invariants are computed based on the information provided by both the shape boundary and its interior region. The moments used to construct the moment invariants are defined in the continuous space as

\[ M_{pq} = \int \int x^p y^q f(x,y) \, dx \, dy \] \hspace{1cm} [5.1]

\( M_{pq} \) is the two-dimensional moment of the function \( f(x,y) \). The order of the moment is \( (p + q) \) where \( p \) and \( q \) are both natural numbers. For implementation in digital form this becomes:

\[ M_{pq} = \sum_x \sum_y x^p y^q f(x,y) \] \hspace{1cm} [5.2]

To normalise for translation in the image plane, the image centroids are used to define the central moments. The co-ordinates of the centre of gravity of the image are calculated using equation (2) and are given by:
\[
\bar{x} = \frac{M_{10}}{M_{00}} \quad \quad \quad \quad \bar{y} = \frac{M_{01}}{M_{00}} \quad \quad [5.3]
\]

The central moments can then be defined in their discrete representation as:

\[
\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p (y - \bar{y})^q \quad [5.4]
\]

The moments are further normalised for the effects of change of scale using the following formula:

\[
\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^\gamma} \quad [5.5]
\]

Where the normalisation factor: \( \gamma = (p + q / 2) + 1 \). From the normalised central moments a set of seven values can be calculated and are defined by:

\[
\begin{align*}
\phi_1 &= \eta_{20} + \eta_{02} \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (\eta_{03} - 3\eta_{21})^2 \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{03} + \eta_{21})^2 \\
\phi_5 &= (3\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 \\
&\quad - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \\
&\quad \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{align*}
\]
\[ \phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \]
\[ + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \]
\[ \phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 \]
\[ - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03}) \]
\[ \times [3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{30})^2] \]

These seven invariant moments, \( \phi_i, 1 \leq i \leq 7 \), were additionally shown to be independent of rotation. However they are computed over the shape boundary and its interior region and this set of invariant moments makes a useful feature vector, \( \phi_i \), for the recognition of objects which must be detected regardless of the position, size or orientation.

5.1.2 Complex Radial Moments

The term complex moments is used to encompass all classes of moment functions which have complex kernels. Commonly used functions of this type are radial moments, Fourier-Mellin descriptors and Zernike moments. In addition, a generalization of geometric moments defined in the complex plane is also specifically referred in literature as complex moments.

Radial moments are defined using polar coordinate representation of the image space. The main advantage with the radial moments is that the image rotations can be directly translated into the corresponding moment transforms. The equations of Radial moments invariants expressed in terms of geometric moments are given as following:
5.1.3 Zernike Moments

Zernike moments were first introduced by Teague [46] based on the orthogonal functions called Zernike polynomials. Though computationally very complex compared to geometric and Legendre moments, Zernike moments have proved to be superior in terms of their feature representation capability and low noise sensitivity.

The Zernike moments are directly related to the central moments $\mu_{pq}$. The equations of Zernike moments invariants expressed in terms of geometric moments are given as[47] following:

\[
R1 = \mu_{20} + \mu_{02};
\]

\[
R2 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2;
\]

\[
R3 = (\mu_{20} + \mu_{02})^2 + 4\mu_{11};
\]

\[
R4 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2; \quad [5.7]
\]

\[
Z1 = 3\{2(\mu_{20} + \mu_{02}) - 1\}/\pi;
\]

\[
Z2 = 9\{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2\}/\pi;
\]

\[
Z3 = 16\{(\mu_{03} - \mu_{21})^2 + (\mu_{30} - 3\mu_{12})^2\}/\pi^2;
\]

\[
Z4 = 144\{(\mu_{03} + \mu_{21})^2 + (\mu_{30} + 3\mu_{12})^2\}/\pi^2;
\]
These Hu moments, Zernike moments and complex moments are calculated for each training image to form the feature vectors, \( \Phi Z \) and \( C \), used to construct the corresponding library for testing.

5.2 Scale Invariant Feature Transform (SIFT) Based object Detection

Scale-invariant feature transform (or SIFT) proposed by David Lowe in 2003\([48]\) is an algorithm for extracting distinctive features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale, rotation, and partially invariant (i.e. robust) to change in 3D viewpoint, addition of noise, and change in illumination. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched
with high probability against a large database of features, providing a basis for object recognition.

For object matching, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors.

5.2.1. Construction of a Scale-Space Representation

SIFT uses a method called scale-space representation. A set of scales are created from the original image by means of resizing the image along both axes by the factor of two, thus creating an image that is one-quarter the size of the original one. This step is then repeated until the image is too small to shrink it any more (in Lowe’s paper a this step is repeated five times). These resized images (and the original as well) are convoluted with a Gaussian kernel, thus blurring the image. This blurring is also repeated several times. A set of blurred images is called an octave, so the whole scale-space has $n$ octaves, where $n$ is the number of the different sized images. Figure 5.1 shows an example for a scale-space representation of an image of UAV. The image in the top left corner is the original. In each row, five blurred images can be seen. The images in the second, third and fourth row are shrinked down, and of course the Gaussian blurring is applied to them as well.
After these images are ready, the keypoint candidates can be found in the so called Difference-of-Gaussian images. These DoG images need to be calculated first. It is done by taking two neighboring images in an octave, and substracting the more blurred from the less blurred, as shown in Figure 5.2. This calculation results in images that can be seen in Figure 5.3.

5.2.2 Local Extrema Detection

The algorithm then looks for keypoint candidates in these DoG images. For this, the grey level of a pixel in a DoG image is compared to it’s neighbors. In this case, neighboring pixels mean the 8 neighbors in the same image, and 18 pixels from the images of the neighboring scales,
resized to the same size (see Figure 5.4.). The point is selected as a keypoint candidate if and only if it’s gray value is greater or smaller than of all those in it’s neighborhood.

This method usually results in a vast number of keypoint candidates. On one hand it good, because the more keypoints there are, the more robust the recognition is. This however only applies if those keypoints are stable (meaning on a slightly different image, that was for example rotated, taken under different illumination conditions or from a different camera angle, the same point will be found at the same positions).
Unfortunately a large number of the keypoint candidates are worthless for recognition. The reason for having unstable keypoints are twofold: low contrast parts and points close to an edge.

![Figure 5.3: Difference-of-Gaussian example](image)

5.2.3 Removal of Unstable Keypoints Along the Edges

The problem with points around the edges is that the Difference-of-Gaussian function has a strong response along them, even if the edge is poorly determined, which makes the points unstable even to small amounts of noise. Of course it is highly unwanted, and that’s why these points need to be identified and removed from the set of keypoint candidates. The goodness of a keypoint candidate in this respect can be estimated by looking at the principal curvature values at the point. A poorly defined peak in the Difference-of-Gaussian image has large
principal curvature across the edge, but a small one in the perpendicular direction. Principal curvature can be calculated from the 2*2 Hessian matrix taken at the scale and location of the keypoint candidate:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

[5.9]

The derivatives in this matrix are the differences of neighboring sample points. It can be shown that the principal curvatures of $D$ are proportional to the eigenvalues of $H$. Since only the ratio of the principal curvatures is needed, the eigenvalues do not need to be calculated, and their ratio can be computed using the following formula:

Figure 5.4. Selecting local extremas
Where

\[
\frac{Tr(H)^2}{Det(H)} < r
\]  

[H.10]

H is a 2x2 Hessen matrix, calculated at the location and scale of the keypoint,

Tr(H) is the trace of H

Det(H) is the determinant of H

r is the ratio of the principal curvatures.

After experimenting with different values for r, Lowe and David in [49] found 10 as a good value for r. In practice it means that the keypoint candidates are removed where this value is greater than 10 (if the principal curvature along one direction is more than ten times as high as that of the other direction).

5.2.4. Orientation Assignment

In order to achieve rotation and scale invariance, each keypoint is assigned a base orientation and scale, which is constant even if the image (and thus the keypoint) is transformed or resized. The scale is used to select the Gaussian blurred image so that the computations are going to be scale-invariant. For each image point in image L the respective magnitude and orientation is computed.
After that the gradients around a keypoint are taken into consideration. An orientation histogram is created in which the gradients are stored, according to their angle, as shown in the left side of Figure 5.5. The magnitude of these gradient orientations are also used, they are used as a length of the vectors that mean the orientation of the gradients.

\[ m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \]  \[ 5.11 \]

\[ \theta(x, y) = \tan^{-1}\left(\frac{L(x, y + 1) - L(x, y - 1)}{L(x + 1, y) - L(x - 1, y)}\right) \]  \[ 5.12 \]

Figure 5.5: Assigning orientations and magnitudes in orientation histogram
5.2.5. Keypoint Descriptor Construction

A unique descriptor has to be created for each keypoint, that is to be stored in the database. Figure 23 shows how the mentioned gradient neighborhood of the keypoint is turned into a keypoint descriptor. In this example, 8x8 samples are taken (the keypoint is in the middle), which are divided into 4 4x4 squares (left side). The arrows in the little squares mean the orientation and magnitude of the gradient at that point. These vectors are put into an orientation histogram, which has 8 bins (8 angles as can be seen in one of the four squares on the right side of Figure 23). The vectors are assigned a bin which is closest to their respective angle. Of course if the angles are not the same (the angle of the bin and the angle of the vector), then an additional calculation is used to determine the length of the vector along the direction of the bin. When two vectors go into a bin, then their magnitudes add up. The use of 4x4 squares allow for some position shift among the gradients. The blue circle around the gradients in Figure 23 is a Gaussian weighting function, that is encorporated so that the points closer to the keypoint will be more significant.

This method is also rotation invariant. If the image is rotated, then the bins in the descriptors are also rotated, so for example in one case the bin with the highest number in it points to the north, and in the rotated case it points to the west. Rotation invariance comes from the fact that even if the direction of biggest bin is different, their respective order does
not change, ie. the one to the right of the biggest bin has the smallest number, and the one next to it is the third and so on.

A keypoint descriptors is going to be a vector with n elements, or a point in an n dimensional space.

\[ n = m \times m \times k \]  

where:

m is the number of the subsquares in the keypoint descriptor squares, along one side

k is the number of bins in the orientation histograms.

After excessive testing, Lowe and David found that the best result can be achieved using a 4x4 descriptor created from a 16x16 neighborhood, which results in a 4x4x8=128 element vector as a keypoint descriptor. As a last step the vector is normalized to make it somewhat invariant to illumination change. There are two kinds of problems with illumination that can be avoided: contrast change, where all the image points are multiplied by a constant value, and brightness change where a constant value is added to each of the image points. The effects of contrast change can easily be avoided by normalizing this vector, and the brightness does not have an effect on the gradients because those are calculated from the grey level difference between pixels. There is however a third problem with illumination, and it comes from having 3D objects in the image,
then position change of the object or the light source yields non-linear illumination change. The only way to balance that is to weight down the influence of magnitudes, because non-linear lighting change affects the magnitudes more than the orientation.

5.2.6. Object Recognition

The method presented above has to be performed when training the database and when the database is ready and an object needs to be identified. The training phase is the simpler case, the keypoints are inserted in the database along with information about the object which it is a part of. After the teaching phase, when the user wants to identify an object, the keypoint descriptors have to be compared to the previously stored descriptors. However due to high dimension of the descriptors and the fact that a the method needs to be reliable some problems arouse that have to be solved.

5.2.7. Keypoint Matching

To find a matching keypoint descriptor in a database, it’s nearest neighbor has to be found in the n dimensional space. This is not hard in theory, however it can result in false matches, because if the image point is not in the database (this usually happens when parts of the background in an image are taken as keypoints), than the nearest neighbor has nothing to do with that image point. As a solution to this problem, not just the nearest neighbor, but the two nearest neighbors are taken (where the second nearest neighbor is not part of the object of
the first nearest neighbor), and their distances to the given point are measured, and then it is possible to guess the correctness of a match from the ratio of these distances. The idea behind this is that keypoint descriptors of an object will be close to each other compared to descriptors of different objects, and that means that if a point is part of an object it should be significantly closer to the keypoint descriptors of the same object (the nearest neighbor), than to any other descriptor of an other object (the second nearest neighbor). If these distances are roughly equal, there is a high chance that the given point is not part of either of these objects. In [49] 0.2 is used as a distance ratio, so only those points are kept that lay at least five times as close to the nearest group than to the second nearest. According to some test results, this method identifies 90% of the false matches while discarding only 5% of correct matches.

5.2.8. Nearest Neighbors

Finding the nearest neighbor of a keypoint descriptor is a challenge in itself. The problem is that for high dimensional spaces the convention nearest neighbor finder algorithms do not work faster than calculating the Euclidean distance between the given point and every other points, so Lowe suggests using an approximation algorithm called Best Bin First. This does not necessarily gives back the nearest neighbor, instead it gives back the point that is with high probability the nearest neighbor. And even if it wrong, the point given back will be close to the nearest
neighbor. It only checks a few hundred possible points close to the given point. More details can be found about the BBF algorithm in [50].

5.2.9. Object Matching

After the false points from the background are discarded, there is still a problem with the points of valid objects that do not belong to the object that has to be found. A model fitting algorithm is required to do this. The problem is that RANSAC and Least Median of Squares do not work reliably if the ratio of inliers and outliers (points belonging to the given object and to other objects) is less than 50%. In a typical object recognition scenario this ratio can easily be 1% inliers with 99% outliers. The keypoints specify 4 parameters: the 2D coordinates, the scale and the rotation of the model. If there are more keypoints from the same object, the scale and rotation attributes will be equal, and the 2D coordinates will be consequent with the coordinates of those keypoints of the training objects in the database. Using these information the keypoints vote for objects they might belong to, and if there are objects with more then 3 votes than those objects are going to be considered found. Affine transform invariance can be achieved by allowing a small error range in the parameter values. Objects with less than three votes are discarded. A final decision is made after probabilistic calculations.

5.2.10. Recognition Examples

To show how efficient object recognition with SIFT can be, here are examples taken. Figure 5.6.a shows a scene with same UAV object, the
number of keypoints found are 157 for both the cases and all the keypoints mapped correctly as shown in the figure. Figure 5.6.b and 5.6.c shows a scene with two different types of UAV’s, the number of keypoints found in Fig. 5.6.b for object1 and object 2 are 87 and 157 keypoints respectively, only 1 keypoint is matched between the two objects. And the number of keypoints found in Fig. 5.6.c for object1 and object 2 are 157 and 100 keypoints respectively; only 2 keypoints are matched between the two objects.

Figure 5.6.a.
Noise Effect

The number of keypoints found in Fig. 5.7.a. for object1 and object 2 are 157 and 551 keypoints respectively, 82 keypoints are matched between the two objects.

Figure 5.7.a.

The number of keypoints found in Fig. 5.7.b. for object1 and object 2 are 81 and 551 keypoints respectively, zero keypoints are matched between the two objects.

The number of keypoints found in Fig. 5.7.c. for object1 and object 2 are 100 and 551 keypoints respectively; only 4 keypoints are matched between the two objects.
Under Occlusion

The number of keypoints found in Fig. 5.8.a. for object1 and object 2 are 157 and 144 keypoints respectively, 94 keypoints are matched between the two objects.

Figure 5.8.a.

The number of keypoints found in Fig. 5.8.b. for object1 and object 2 are 81 and 144 keypoints respectively; zero keypoints are matched between the two objects.
The number of keypoints found in Fig. 5.8.c for object1 and object 2 are 100 and 144 keypoints respectively; zero keypoints are matched between the two objects.
5.3. Affine- Scale Invariant Feature Transform

SIFT is fully invariant with respect to only four parameters namely zoom, rotation and translation, the ASIFT method [51] treats the two left over parameters: the angles defining the camera axis orientation.

Against any prognosis, simulating all views depending on these two parameters is feasible. The method permits to reliably identify features that have undergone very large affine distortions measured by a new parameter, the transition tilt.

5.3.1. When does it work?

The SIFT method works to compare 2D objects or 3D objects with flat enough details, taken from similar view angles but at arbitrary distances.

- The typical failure cases are:
- The illumination conditions are different (for instance daylight/nightlight).
- The object has a reflecting surface (typically cars mirrors; they change completely aspect under different view angles).
- The object has a strong 3D structure: in that case a change of view angle alters drastically its aspect.
- The object has a self similar or periodic structure: then “true” mismatches occur.
- The view angle is too different.

ASIFT corrects the last problem: if the object is under view has similar illumination conditions, has rather flat surface, and is not a mirror, then
ASIFT retrieves the object even under extreme changes of angle. In technical terms, ASIFT is more affine invariant than SIFT.

5.3.2 The ASIFT Algorithm

The idea of combining simulation and normalization is the main successful ingredient of the SIFT method. Indeed, scale changes amount to blur and cannot be normalized. Thus SIFT normalizes rotations and translations, but simulates all zooms out. David Pritchard’s extension of SIFT [52] simulated four additional tilts. This is actually a first step toward the algorithm described below, which is also summarized in Fig. 5.9.

![ASIFT Diagram](image)

**Fig. 5.9. Overview of ASIFT.**

Many pairs of rotated and tilted images obtained from images A and B are compared by SIFT.
1. Each image is transformed by simulating all possible affine distortions caused by the change of camera optical axis orientation from a frontal position. These distortions depend upon two parameters: the longitude $\Phi$ and the latitude $\Theta$. The images undergo $\Phi$-rotations followed by tilts with parameter $t = \frac{1}{\cos \Phi}$ (a tilt by $t$ in the direction of $x$ is the operation $u(x, y) \rightarrow u(tx, y)$). For digital images, the tilt is performed by a directional $t$-subsampling. It requires the previous application of an antialiasing filter in the direction of $x$, namely the convolution by a Gaussian with standard deviation $c\sqrt{t^2-1}$. The value $c = 0.8$ is the value chosen by Lowe for the SIFT method [49]. As shown in [53], it ensures a very small aliasing error.

2. These rotations and tilts are performed for a finite and small number of latitude and longitude angles, the sampling steps of these parameters ensuring that the simulated images keep close to any other possible view generated by other values of $\Phi$ and $\Theta$.

3. All simulated images are compared to each other by some scale invariant, rotation invariant, and translation invariant algorithm (typically SIFT). Since SIFT normalizes the translation of the camera parallel to its focal plane and the rotation of the camera around its optical axis, but simulates the scale change, all six camera parameters are either normalized or simulated by ASIFT.
4. The simulated latitudes $\Theta$ correspond to tilts $t = 1, a, a^2, \ldots, a^n$, with $a > 1$. Taking $a = \sqrt{2}$ is a good compromise between accuracy and sparsity. The value $n$ can go up to 5 or more. That way, all transition tilts from 1 to 32 and more are explored.

5. The longitudes $\Phi$ follow for each $t$ an arithmetic series $0, b/t, \ldots, kb/t$ where $b = 72^\circ$ is a good compromise and $k$ is the last integer such that $kb/t < 180^\circ$.

6. Complexity: Each tilt is a $t$ sub-sampling dividing the image area by $t$. The number of rotated images for each tilt is $(180/72)t = 2.5t$. Thus, the method complexity is proportional to the number of tilts. Controlling the total area of the simulated images is equivalent to controlling the algorithm complexity. Indeed, the SIFT search time and memory size are proportional to the image area. This complexity can be further downgraded by a) subsampling the query and search images; b) identifying the successful pairs ($t, \Phi$); c) going back to the original resolution only for these pairs.

7. This description ends with a concrete example of how the multi-resolution search strategy can actually make the algorithm only twice slower than SIFT. Take $a = \sqrt{2}$, $n = 5$. The maximal absolute tilt for each image is 5.7 and the maximal transition tilt goes up to 32. The simulated image area is $5 \times 2.5 = 12.5$ times the original area. By a $3 \times 3$-subsampling of the original, this area is reduced to 1.4 times the one of the original image. If this reduction is applied to both the query and the
search image, the overall comparison complexity is equivalent to twice the SIFT complexity. Fig. 5.10 shows the relatively sparse sampling of the longitude-latitude sphere needed to perform a fully affine recognition.

A mathematical proof that ASIFT is fully affine invariant (up to obvious precision issues) is given in [54].

5.3.3. Recognition examples

To show how effective object recognition with ASIFT can be, here are two examples taken. Figure 5.11.a. shows a scene with same UAV object, the number of keypoints found are 333 for both the cases and all the keypoints mapped correctly as shown in the figure. Figure 5.11.b. and 5.11.c. shows a scene with two different types of UAV's, the number of keypoints found in Fig. 5.11.b. for object 1 and object 2, the number of keypoints found are 183 and 333 respectively, none of keypoint is

Fig. 5.10. Sampling (block dots) of the parameters $\Theta = \arccos \frac{1}{t}$ and $\Phi$ in a zenith view of the observation half sphere.
detected between the two objects. And the number of keypoints found in Fig. 5.11.c. for object 1 and object 2 are 288 and 333 keypoints respectively; none of the keypoint are detected between the two objects.

Figure 5.11.a.

Figure 5.11.b.
Noise Effect

The number of keypoints found in Fig. 5.12.a. for object 1 and object 2 are 333 and 434 keypoints respectively, 101 keypoints are matched between the two objects.
The number of keypoints found in Fig. 5.12.b. for object1 and object 2 are 183 and 434 keypoints respectively, zero keypoints are matched between the two objects.

Figure 5.12.b.

The number of keypoints found in Fig. 5.12.c for object1 and object 2 are 288 and 434 keypoints respectively; none of the keypoint are detected between the two objects.

Figure 5.12.c.
Under Occlusion

The number of keypoints found in Fig. 5.13.a. for object1 and object 2 are 333 and 356 keypoints respectively, 113 keypoints are matched between the two objects.

Figure 5.13.a.

The number of keypoints found in Fig. 5.13.b. for object1 and object 2 are 183 and 356 keypoints respectively; zero keypoints are matched between the two objects.

The number of keypoints found in Fig. 5.13.c. for object 1 and object 2 are 288 and 356 keypoints respectively; zero keypoints are matched between the two objects.
5.4. Object Detection

We have the 3D modeling software to create two types of databases. Database1 contains 216 images of objects (800x600) of different rotational angles. The database contains the possible combinations of
Pitch, Yaw, and Roll angles from 0~40, with a minimal step size of 4 for each angle. In another word, there are six possible values: 0, 4, 8, 12, 16, 20 for pitch, Yaw, and Roll, hence a total of 216 training images.

Database 2 contains 35 different aircraft images shown in Fig.5.14 of 5 different views, hence a total of 175 images. Database 1 and Database 2 are used for pose estimation and identification respectively. Each lookup table entry corresponds to one of the images in the database. For each lookup table entry, the index value of the object, the values of absolute moment invariants, the 2D rotational angle θ from the object’s principle axes, and the scaling factor of the corresponding database image are tabulated.
Given two sets of descriptors, how do we measure their degree of similarity? An appropriate classification is necessary if unknown shapes are to be compared to a library of known shapes. If two shapes, A and B, produce a set of values represented by $a(i)$ and $b(i)$ then the distance between them can be simply measured by Euclidean Distance (ED)

$$ED = \sqrt{\sum_{i=1}^{n}(a(i) - b(i))^2}$$ \hspace{1cm} (9).

Testing

In the first step, the object image is acquired by using the camera mounted on the UAV following the leader UAV, background is
subtracted, binarization of gray images is done. Then, a set of moment invariants is extracted. In the identification phase, an object image is classified by pairwise comparison of extracted features and database values according to the steps listed below:

Identification Algorithm 1

1. Image acquisition
2. A closed contour of the aircraft is extracted by background subtraction, binarization.
3. Three set of moments (Hu, complex and Zernike moments) of order three are calculated using equations 6, 7, 8.
4. The centre of gravity of the image is found
5. The central moment, which eliminates the translation effect, is found.
6. Normalization is done for the moments by dividing them with the $m_{00}$ (size of the image) for removing the scaling effect.
7. The respective invariant moments are calculated
8. For a given test image, its moment invariants are compared with all the model objects moment invariants using the ED in (9).
9. The model object with a least Euclidean distance is a recognized object.

Identification Algorithm 2

1. Image acquisition
2. A closed contour of the aircraft is extracted by background subtraction, binarization.

3. The features (SIFT, ASIFT) are calculated.

4. For a given test image, its features are compared with all the model objects features.

5. The model object with a high feature mapping is a recognized object.

The following Tests are performed

Test 1: The performance of the various moment invariants for different orientations is studied using the data samples. A total of 100 test images have been taken. Example includes the following images for the same object.
Test 2

The performance of moment invariants is evaluated in this section for the classification of various types under different scales.

The scale is varied from 0.1 to 1.5 of the original images. We randomly chose 300 images from the database for identification.

Test 3

To study the effect of the noise, we choose Gaussian, salt and pepper, Poisson and speckle type.

a) Three different aircraft images have taken from the database and a Gaussian additive noise of zero mean and variance of 0.01 to 0.20 with
an increment of 0.01 has been added to the clean image. A total of 300 test images have been chosen for identification.

b) Three different aircraft images have been taken from the database and a salt and pepper noise with a noise density $D$. This affects approximately $D^*$ number of pixels in the original image. The value of $D$ varied from 0.01 to 0.20 with an increment of 0.01 has been added to the original image which is considered not noisy.
c) Three different aircraft images have taken from the database and a speckle noise adds multiplicative noise to the image I, using the equation $J = I + n^*I$, where $n$ is uniformly distributed random noise with mean 0 and variance $V$. $V$ varies from 0.01 to 0.20 with an increment of 0.01. It has been added to the original image. A total of 300 test images are used for identification.
d) Three different aircraft images are taken from the database and a Poisson noise was generated from the data.

For example, if a pixel in a 8-bit pixel and the input has the value 10, then the corresponding output pixel will be generated from a Poisson distribution with mean 10.

Test 4

The performance of the various moment invariants in the presence of motion blur is studied using the aircraft data samples. Motion blur images are generated by varying the linear motion of a camera by LEN pixels, with an angle of THETA degrees in a counter-clockwise direction. Three different aircraft images are taken from the database and a motion blur of LEN 1 to 21 with an increment 2 and THETA value of 1 to 21 with an increment of 5 has been added to image.
Test 5

The performance of the various moment invariants in the presence of occlusion is studied using the aircraft data samples. The occlusion is roughly created by erasing a part of the objects image. Three different aircraft images have taken from the database and a total of 300 test images are generated for identification purpose. A minimum of 5% and a maximum 50 % of occlusion have been applied. Some examples are shown below
The tables below show the accuracy of identification under different conditions.
Table 5.1 Average identification accuracy (%) by different sets of moment invariants with respect to different type of distortions and scale

<table>
<thead>
<tr>
<th>Type</th>
<th>Scale</th>
<th>Gaussian Noise</th>
<th>Salt and Pepper Noise</th>
<th>Poisson Noise</th>
<th>Speckle Noise</th>
<th>Motion Blur</th>
<th>Occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu Moments</td>
<td>98</td>
<td>62</td>
<td>39.5</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>42</td>
</tr>
<tr>
<td>Complex Radial Moments</td>
<td>99</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>96</td>
<td>100</td>
<td>48</td>
</tr>
<tr>
<td>Zernike Moments</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>100</td>
<td>57.5</td>
</tr>
<tr>
<td>SIFT</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>ASIFT</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Table 5.2 Identification accuracy (%) by different sets of moment invariants with respect to scale change

<table>
<thead>
<tr>
<th>Scale</th>
<th>Hu moments (%)</th>
<th>Complex moments (%)</th>
<th>Zernike moments (%)</th>
<th>SIFT (%)</th>
<th>ASIFT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>60</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.2</td>
<td>90</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.3</td>
<td>90</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.7</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.8</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.9</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1.1</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1.2</td>
<td>100</td>
<td>100</td>
<td>90</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1.3</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1.4</td>
<td>80</td>
<td>90</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1.5</td>
<td>80</td>
<td>80</td>
<td>70</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

For the scale of 0.4 to 1.1, all the algorithms works perfect.
Table 5.3 Identification accuracy (%) by different sets of moment invariants with respect to Gaussian noise

<table>
<thead>
<tr>
<th>Gaussian noise</th>
<th>Hu moments (%)</th>
<th>Complex moments (%)</th>
<th>Zernike moments (%)</th>
<th>SIFT (%)</th>
<th>ASIFT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.02</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.03</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.04</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.05</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.06</td>
<td>94.4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.07</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.08</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.09</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.1</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.11</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.12</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.13</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.14</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.15</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.16</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.17</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.18</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.19</td>
<td>0</td>
<td>94.4</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
Table 5.4 Identification accuracy (%) by different sets of moment invariants with respect to Salt and Pepper noise

<table>
<thead>
<tr>
<th>Salt and pepper (%)</th>
<th>Hu moments (%)</th>
<th>complex moments (%)</th>
<th>zernike moments (%)</th>
<th>SIFT (%)</th>
<th>ASIFT (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.02</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.03</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.04</td>
<td>94.4</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.05</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.06</td>
<td>86.6</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.07</td>
<td>78.8</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.08</td>
<td>78.8</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.09</td>
<td>78.8</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.1</td>
<td>40</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.11</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.12</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.13</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.14</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.15</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.16</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.17</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.18</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.19</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
The following conclusions can be drawn

- ASIFT and SIFT are more robust to noise and scale changes and recognizes the object 100% even under 50% occlusion.
- Among the moments:
  - Zernike moments are noise immune, but degrade the performance with the scale increase
  - Complex radial functions are also noise immune, except very high noise levels
  - Complex radial moments are less computationally extensive compared to Zernike’s set. They are also a better choice for scale invariant identification.
  - Zernike’s moments are robust under partial occlusions.

5.5 Moment Functions for Pose Identification

The pose estimation is performed based on the following procedures. We populate a DB 1 (a look up table) with the moments for a fixed one angle and two other varying per object. Based on the moment values calculated from an entry image, we calculate the angle $\phi$ according to (10) below and Table 5.5

$$\phi = \frac{1}{2} \tan^{-1} \left( \frac{2\mu_{11}}{u_{20} - \mu_{02}} \right)$$

(10)
Table 5.5 Angle calculation from moment functions.

<table>
<thead>
<tr>
<th>$u_{20} - \mu_{02}$</th>
<th>$\mu_{11}$</th>
<th>$\emptyset$</th>
<th>$\varepsilon = \frac{2\mu_{11}}{u_{20} - \mu_{02}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>Zero</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>Positive</td>
<td>+45°</td>
<td></td>
</tr>
<tr>
<td>Zero</td>
<td>Negative</td>
<td>−45°</td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td>Zero</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>Zero</td>
<td>−90°</td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td>Positive</td>
<td>$\frac{1}{2} \text{tan}^{-1}(\varepsilon)$</td>
<td>$0 &lt; \emptyset &lt; 45°$</td>
</tr>
<tr>
<td>Positive</td>
<td>Negative</td>
<td>$\frac{1}{2} \text{tan}^{-1}(\varepsilon)$</td>
<td>$-45° &lt; \emptyset &lt; 0$</td>
</tr>
<tr>
<td>Negative</td>
<td>Positive</td>
<td>$\frac{1}{2} \text{tan}^{-1}(\varepsilon) + 90°$</td>
<td>$45° &lt; \emptyset &lt; 90°$</td>
</tr>
<tr>
<td>Negative</td>
<td>Negative</td>
<td>$\frac{1}{2} \text{tan}^{-1}(\varepsilon) - 90°$</td>
<td>$-90° &lt; \emptyset &lt; -45°$</td>
</tr>
</tbody>
</table>

and use the moments multiplied by angle as an index to the look up table. Then, we calculate distances between moments per corresponding entry. The scale is calculated as
where \( m'_{00} \) is the area of the object in the test image, and \( m_{00} \) is the area of the object in the lookup table image.

Calculation of the control parameters, such as a linear speed and pitch, yaw and roll angles changes is out of the scope of this work.

The performance of the moment invariants is evaluated in this section for the identification of the pose under different conditions.

Test 1

The performance of the various invariants is evaluated in this section for the identification of pose of the aircraft under different scale.

The scale is varied from 0.1 to 1.5 of original image. We randomly chose 40 images from the database for identification.

Test 2

a) To study the effect of the noise, we choose several types of noise like Gaussian, salt and pepper, Poisson and speckle using the aircraft data samples.

b) A Gaussian additive noise of zero mean and variance of 0.01 to 0.20 with an increment of 0.01 has been added to the clean image. A total of 40 test images have been chosen for identification of pose.
c) A salt and pepper noise with a noise density D. This affects approximately D* number of pixels in the original image. The value of D varied from 0.01 to 0.20 with an increment of 0.01 has been added to the original image which is considered not noisy. A total of 40 test images have been chosen for identification of pose.

d) A speckle noise adds multiplicative noise to the image I, using the equation \( J = I + n*I \), where \( n \) is uniformly distributed random noise with mean 0 and variance V. V varies from 0.01 to 0.20 with an increment of 0.01. It has been added to the original image. Therefore a total of 40 test images have been chosen for identification of pose.

e) A Poisson noise was generated from the data. For example, if a pixel in a 8-bit pixel and the input has the value 10, then the corresponding output pixel will be generated from a Poisson distribution with mean 10. A total of 30 test images chose for pose estimate.

Test 3

The performance of the various moment invariants in the presence of motion blur is studied using the aircraft data samples. Motion blur images are generated by varying the linear motion of a camera by LEN pixels, with an angle of THETA degrees in a counter-clockwise direction. Three different aircraft images are taken from the database and a motion blur of LEN 1 to 21 with an increment 2 and THETA value of 1 to 2
with an increment of 5 has been added to the clean image. A total of 40 test images chosen randomly for identification of pose.

Test 4

The performance of the various moment invariants in the presence of occlusion is studied using the aircraft data samples. The occlusion is roughly created by erasing a part of the objects image. A total of 40 test images have been chosen for pose estimation with a minimum of 5% and a maximum 50 % degree of occlusion images have been taken

Table 5.6 average accuracy (%) under different distortions

<table>
<thead>
<tr>
<th>Moments Type</th>
<th>Scale</th>
<th>Gaussian Noise</th>
<th>Salt and Pepper Noise</th>
<th>Poisson Noise</th>
<th>Speckle Noise</th>
<th>Motion Blur</th>
<th>Occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>80</td>
<td>55.5</td>
<td>46</td>
<td>94</td>
<td>75</td>
<td>72.5</td>
<td>48.5</td>
</tr>
<tr>
<td>Complex Radial</td>
<td>86</td>
<td>82</td>
<td>74.5</td>
<td>85</td>
<td>78</td>
<td>92</td>
<td>54</td>
</tr>
<tr>
<td>Zernike</td>
<td>91.5</td>
<td>86</td>
<td>92</td>
<td>91</td>
<td>86</td>
<td>87</td>
<td>57</td>
</tr>
</tbody>
</table>
Tables 5.7–5.8 tabulate parameters of erroneous estimates.

**Table 5.7 Accuracy vs scale**

<table>
<thead>
<tr>
<th>Moments type</th>
<th>Wrong Estimated Scales of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>0.1, 0.2, 0.3, 0.4, 0.5, 1.4, 1.5</td>
</tr>
<tr>
<td>Complex</td>
<td>0.1, 0.2, 0.3, 0.4, 1.5</td>
</tr>
<tr>
<td>Zernike</td>
<td>0.1, 0.2, 1.5</td>
</tr>
</tbody>
</table>

**Table 5.8 Accuracy vs Gaussian noise**

<table>
<thead>
<tr>
<th>Moments type</th>
<th>Wrongly Estimated for variance of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>0.12, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Complex</td>
<td>0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Zernike</td>
<td>0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
</tbody>
</table>
Table 5.9 Accuracy vs Poisson noise

<table>
<thead>
<tr>
<th>Moments</th>
<th>Wrongly Estimated for noise density D of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Complex</td>
<td>0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Zernike</td>
<td>0.17, 0.18, 0.19, 0.20</td>
</tr>
</tbody>
</table>

Table 5.10 Accuracy vs speckle noise:

<table>
<thead>
<tr>
<th>Moments</th>
<th>Wrongly Estimated for variance of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Complex</td>
<td>0.16, 0.17, 0.18, 0.19, 0.20</td>
</tr>
<tr>
<td>Zernike</td>
<td>0.17, 0.18, 0.19, 0.20</td>
</tr>
</tbody>
</table>

Table 5.11 Accuracy vs occlusion:

<table>
<thead>
<tr>
<th>Moments</th>
<th>Wrongly Estimated for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hu</td>
<td>Above 7% occlusion</td>
</tr>
<tr>
<td>Complex</td>
<td>Above 10% occlusion</td>
</tr>
<tr>
<td>Zernike</td>
<td>Above 15% occlusion</td>
</tr>
</tbody>
</table>
By all the tests, Zernike moments provide a highest accuracy. Although 100 % of the accuracy is not attained under different image distortions applicable to a realistic environments, if there is no an abrupt scale change, the approach allows for sustaining a coordinated flight. Under a moderate speed/path change, a smooth tracking and pose adjustment can be attained, thus the scale related inaccuracy can be eliminated.

Conclusion
Various methods have been selected as candidates for object detection and pose estimation and studied experimentally. The results show that features obtained by ASIFT are more robust to rotation, scale, noise and partial occlusions, and thus a high quality recognition is attained. Computationally, they are more intensive compared to invariant moments. For pose estimation, three different moment functions have been implemented. Of all the methods the Zernike moments yield a highest accuracy under all environments. For real time implementation, Zernike moments are good choice for both identification and pose estimation.
CHAPTER 6

HARDWARE IMPLEMENTATION

In this chapter we re-design the system to “fit it” to on-board implementation. A real-time hardware system is developed for background subtraction, detection, tracking and pose estimation. First, the techniques utilized for hardware implementation are discussed, then an overview on TMS320C64x DSP Family is given. Thirdly, the experimental setup and block diagram of the algorithm is presented. Finally, the system characteristics are discussed in comparison to modules running on software platforms. Specifically we compare the total computation/running times. We have modified a certain part of the algorithms due to the inefficiencies associated with manually translating MATLAB “concept” code into C code for implementation.

6.1. Embedded MATLAB

Embedded MATLAB tools can automatically convert a well-defined subset of MATLAB language, called Embedded MATLAB, into embeddable C code. This technology can reduce the development and verification cost of manual translation from MATLAB to C. The Embedded MATLAB language subset supports more than 270 MATLAB operators and functions and 90 Fixed-Point Toolbox functions.
Converting a typical MATLAB algorithm into embeddable C code involves accommodating several implementation requirements:

a) **Data Type Management**—Data types must be determined before implementation. For example, the pixel values in image processing are often represented as 8-bit unsigned integers, and samples in speech processing are represented as 16-bit signed integers. The use of default 64-bit double precision variables in MATLAB is not memory efficient.

b) **Static Memory Allocation**—MATLAB seamlessly handles dynamically changing variable sizes at run time. In embedded applications, however, we usually avoid dynamic memory allocation and statically define memory for a given size and data type before using it.

c) **Reduction of Computational Complexity and Memory Footprint**—embedded software designers spend a lot of time mapping high-level algorithms to operate efficiently within the limited memory and computational resources of the target hardware. This effort results in tuning the design to the instruction set and data representation of the target processor.

d) **Fixed-Point Support**—Implementation in embedded software or hardware may require that the algorithm be completely specified with fixed-point data types.

Typically to perform these modifications we first translate the MATLAB algorithm into C code, which creates the design gap mentioned above. Translation may introduce errors or numerical changes into the C code. If those changes are the intentional result of code optimizations,
they need to be reproduced them in the MATLAB algorithm to maintain equivalence. This process adds unnecessary work and potential for errors. A workflow for embedded implementation based on Embedded MATLAB addresses these issues.

Unlike many MATLAB algorithms, Embedded MATLAB code is not an abstract mathematical representation of the design. It contains all the details needed for an efficient, embeddable C implementation. Any MATLAB code that complies with the Embedded MATLAB subset can generate embeddable C code. The process of ensuring compliance with the Embedded MATLAB subset involves the same four implementation requirements discussed previously. With Embedded MATLAB, the implementation constraints are specified directly in the MATLAB code.

Applying the implementation requirements allows for:

- Maintaining one copy of your design in MATLAB
- Elaborating the design from a concept form to an implementation-ready form by incorporating embedded design constraints directly in MATLAB
- Iterating, test, and debug the code using the visualization and analysis capabilities available in the MATLAB environment
- Verifying the functional correctness of the elaborated design
- Automatically generate embeddable C code using Real-Time Workshop

Making a MATLAB algorithm compliant with the Embedded MATLAB subset may require the following steps to perform:
• **Set data types for variables**—assign data types either in the body of the MATLAB function or at compile time. Assignment at compile time is more convenient, since it permits a single MATLAB function to produce multiple C function variants with different data types, dimensions, and complexity. Because we specify data type and sizes of variables at the function interface, the data types and sizes of variables used in the body of the function are automatically inferred.

• **Accommodate array size changes without dynamic data allocation**—In MATLAB, the size of a variable may change between loop iterations. We can accommodate array size changes for embedded implementations by using buffers of a constant maximum size and addressing sub-portions of the constant-size buffers.

• **Create an Embedded MATLAB compliant function**—Not all MATLAB toolbox functions comply with the Embedded MATLAB subset. These functions are designed for flexibility and numerical accuracy, not for embedded implementation. We can use the toolbox function as a template or reference to develop a functionally equivalent Embedded MATLAB function that meets the computational and memory constraints needed for efficient embedded C-code implementation. The desired C code can then be generated automatically from the Embedded MATLAB code.

• **Convert from floating-point to fixed-point**—We can use tools such as data logging and data-type override in the Fixed Point Toolbox to
observe the dynamic ranges of variables in your MATLAB algorithm. These tools help us to convert the algorithm to an optimized fixed-point representation in MATLAB. Because the original and the converted algorithms are both in MATLAB, you can directly compare the floating- and fixed-point results to ensure that the differences are within acceptable tolerance levels.

6.2 TMS320DM642 Processor

The DM642 Evaluation Module (EVM)[119] is a low-cost standalone development platform that enables users to evaluate and develop applications for the TI C64xx DSP family. The EVM also serves as a hardware reference design for the TMS320DM642. Key features include:

- A Texas Instruments TMS320DM642 DSP operating at 720 MHz.
- Standalone or standard PCI computer slot operation.
- 3 video ports with 2 on board decoders and 1 on board encoder.
- 32 Mbytes of synchronous DRAM.
- On Screen display (OSD) via FPGA.
- 4 Mbytes of non-volatile Flash memory.
- AIC23B stereo codec.
- Ethernet interface.
- Software board configuration through registers implemented in FPGA.
- Configurable boot load options.
• JTAG emulation through on-board external emulator interface.
• 8 user LEDs.
• Single voltage power supply (+5V).
• Expansion connectors for daughter card use.
• Dual UART with RS-232 drivers.

6.2.1 C64x Architecture Overview

TMS320C64x CPU core which is composed of two register files, Register File A and Register File B, in TMS320C64x CPU Core, eight
functional units and two data paths is shown in Figure 6.2[111]. Each of the register files have 32 32-bit general-purpose registers which can be used for arithmetic or conditional operations. There are eight functional units which can be divided into two groups. Each group has the same functionality. There are four different types of functional groups whose names are L, S, M, D, which are also shown in Figure 6.2 [111].

Two register cross paths between two groups of functional registers exist in TMS320C64x. These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side’s register file as shown in Figure 6.3. [111]. This increases orthogonality, thus compiler efficiency.

The C64x supports 32 bit load and store operations. There are four paths to registers: two load and two store paths as shown in Figure 6.4. [111]. In C62x and C67x architectures word or double word need alignment to 32-bit or 64-bit boundaries. However, C64x can access words or double words using non-aligned loads or stores which result in increased parallelism, thus, performance improvement.

6.2.2. Chip Level Features

In Figure 6.5. [112], the block diagram of C64x is shown which composes of two-level memory, Enhanced Direct Memory Access (EDMA) controller, External Memory Interfaces (EMIFs) and peripherals.
6.2.2.1. Memory Structure

As mentioned above, a 600 MHz TMS320C64x DSP offers 4800 MIPS. Fast memory which is directly connected to the CPU (Central Processing Unit) is required to process data at this extremely high rate. However, the increase in memory speed could not catch the increase in processor speed which results in a bandwidth dilemma. Therefore, the memory to which the CPU is connected often becomes a processing bottleneck where a possible solution is caches.
Caches, which lie between the CPU and slower system memory, provide code and data to the CPU at the speed of the processor, while automatically managing the data movement from the slower memory. TMS320C64x has a two level memory structure for program and data as shown in Figure 6.6 [112]. L1P, level one data cache, services data accesses from the CPU. On the other hand, L1D, level one data cache, services program fetches from the CPU. Both the program and the data memory share the second level memory, L2, which services the cache misses from both L1P and L1D.
Figure 6.4. C64x Memory Load and Store Paths

Figure 6.5. TMS320C64x DSP Block Diagram
6.2.2.2. EDMA Controller

All data transfer between the on-chip level-two (L2) memory, external memory, and the device peripherals are performed by Enhanced Direct Memory Access (EDMA) [113]. These data transfers include CPU-initiated and event-triggered transfers, master peripheral accesses, cache servicing, and non-cacheable memory accesses. The EDMA architecture has many features designed to service multiple high-speed data transfers simultaneously.
The C64x EDMA can provide 64 channels for independent data transfer. Both one and two-dimensional transfers are supported. Subframes of an image as well as automatically interleave or de-interleave time-division multiplexed (TDM) digital streams can be transferred using 1-D and 2-D. Byte, word, half-word, and double-word data sizes are supported.

6.2.2.3. External Buses

C64x processors support 3 parallel external buses in order to fulfill the high I/O bandwidth requirements. There are two external memory interfaces (EMIFs) [114] which are EMIFA and EMIFB and one host port interface (HPI). 64-bits wide EMIFA is utilized for direct connection to high speed synchronous memory, whereas 16-bit EMIFB is utilized for external I/O peripherals. The two EMIFs are identical except for their width, allowing for a variety of system designs.

32-bit HPI supports communication interface between other processors of industrial type. In some models of C64x, HPI is replaced by PCI interface. PCI bus supplies interface for PCI devices.

6.2.2.4. General Purpose I/O

The general-purpose input/output (GPIO) peripheral provides pins that can be configured as either inputs or outputs. The state of the input which is reflected in an internal register can be detected when configured as an input. On the other hand, the state of the output can be controlled, when configured as an output. There are a total of 16 GPIO pins some of
which are multiplexed with other device pins. Furthermore, the GPIO peripheral can produce CPU interrupts and EDMA events.

6.2.3. Code Development

Programmable DSPs provide software engineers the tools to reduce time to market along with an optimized solution to the application challenge. Sophisticated and easy to use development tools are necessary in order to focus on innovation, product differentiation, and time to market. Historically, there were two distinct DSP tools: code generation (compilers, assemblers, and linkers) and code analyzing (source code debuggers, and profilers). Since they were distinct there was no automatic sharing of data, requiring the developer to constantly switch between different applications. Today development tools enable quick movement through the DSP-based application design process - from concept, to code/build, through debug analysis, tuning, and on to testing.

Code Composer Studio (CCS) [115] is a development environment designed for the Texas Instruments (TI) high performance digital signal processor (DSP) platforms. CCS has the following capabilities:

- Integrated development environment with editor, debugger, project manager, profiler, etc...
- C Compiler, Assembly Optimizer and Linker
- Instruction Set Simulator
- Real-Time Foundational Software (DSP/BIOS)
• Real-Time Data Exchange Between Host and Target (RTDX)
• Real-Time Analysis and Data Visualization

The development flow of most DSP-based applications consists of four basic phases: Application Design, Code Creation, Debug, and Analyze/Tune. The code development cycle of CCS is illustrated in Figure 6.7. [115].

6.2.4. DSP/BIOS

DSP/BIOS [116] is a kernel where run-time services are provided for developers to build DSP applications and manage application resources. It is designed for applications that require real-time scheduling and synchronization, host-to-target communication, or real-time instrumentation. The DSP/BIOS provides easy-to-use powerful program development tools with the following components:

DSP/BIOS Real-Time Analysis Tools are used together with windows within Code Composer Studio to view the program as it executes on the target in real-time.

DSP/BIOS Application Program Interface (API) which lets the user to utilize C or assembly language functions to access and configure DSP/BIOS functions by calling any of over 150 API functions. The Embedded Target for TI C6000 DSP uses the API to let the user access DSP/BIOS from MATLAB. DSP/BIOS Configuration Tool enables the user to add and configure any and all DSP/BIOS objects that is used to instrument the application. This tool is used to configure interrupt
schedules and handlers, set thread priorities, and configure the memory layout on the DSP. Select and configure the foundation modules and kernel objects required by the application with the DSP/BIOS Configuration Tool.

- Furthermore DSP/BIOS minimizes the memory and CPU requirements on the target in the following ways:
  - All DSP/BIOS objects can be created in the Configuration Tool which reduces code size and optimizes internal data structures.
  - The library is optimized to require the smallest possible number of instruction cycles, with a significant portion implemented in assembly language.
  - Communication between the target and the DSP/BIOS Analysis Tools is performed within the background idle loop. This
ensures that the DSP/BIOS Analysis Tools do not interfere with the program’s tasks.

- Error checking that would increase memory and CPU requirements has been kept to a minimum.

DSP/BIOS also provides preemptive multi-threading. There are several thread types such as hardware interrupts, software interrupts, tasks, idle functions, and periodic functions. The priorities and blocking characteristics of threads can be controlled through the choice of thread types. Structures to support communication and synchronization between threads are also provided.

In Figure 6.8. [116] the components of DSP/BIOS within the program generation and debugging environment of Code Composer Studio are shown which reflects the following sequence:

- Programs are written in C or assembly on the host PC
- The objects which will be used in the program is defined in the Configuration Tool
- Then the program is compiled and linked
- The DSP/BIOS Analysis Tools is used to test the program on the target device from Code Composer Studio while monitoring CPU load, timing, logs, thread execution, etc...
6.2.5. Chip Support Library

The chip support library (CSL) [117] is composed of discrete modules which are built and archived into a library file. CSL is written primarily in C with some assembly language where needed. Each module represents an individual application programming interface (API) and is referred to simply as an API module.

The list of CSL API Modules that are currently available is:

- CACHE cache module
- CSL top-level module
• DAT device independent data copy/fill module
• CHIP chip specific module
• DMA direct memory access module
• EDMA enhanced direct memory access module
• EMIF external memory interface module
• HPI host port interface module
• IRQ interrupt controller module
• MCBSP multi channel buffered serial port module
• PWR power down module
• STDINC standard include module
• TIMER timer module

CSL has a two layer architecture: the top layer is the service layer and the bottom layer is the hardware abstraction layer (HAL). The entire purpose of the HAL is to provide the service layer a symbolic interface into the hardware. On the other hand, the actual APIs are defined in the service layer which is the layer the user interfaces to.

6.2.6. Network Developer’s Kit

The new TCP/IP Network Developer’s Kit (NDK) [118] based on the TMS320C6000 DSP platform is a complete and easy-to-use development environment for integrating TI's TCP/IP stack with DSP applications. TI's TCP/IP stack increases system integration and simplifies the design for embedded systems needing network connectivity by running as an extra duty on the same C6000 DSP as the application. This allows designers to
eliminate a separate network processor and use a more cost effective MACPHY device instead.

- TI's TCP/IP stack has the following features:
- NDK TCP/IP can be configured as client, protocol server or router, by adjusting the stack configuration and selecting the network services.
- Developers can program the applications using DSP/BIOS.
- The system provides nearly all the socket functions.

6.3. The Experimental Setup

Figure 6.9
• The image taken by the camera is fed into the processor.
• The image is then sent to the decoder where we choose the format of the input image.
• This image is then sent to the DM642 processor where the main image processing techniques are performed.
• We have used CC Studio V3 to write the code. This is then interfaced to the processor using JTAG.
• After the image is processed it is sent to the encoder where the format of the output image is chosen.
• The output is then displayed on the monitor.

6.4. Real-time tracking/ pose estimation system

![Diagram of real-time tracking/pose estimation system]

Fig. 6.10. Overview of the real-time tracking/ pose estimation system.
Video Capture

The video is captured by Sony 1/4” IR color CCD camera using DM642 EVM video capture simulink block.

Background Subtraction:

For background subtraction, we used approximated median filter because of less complexity and it handles the changing light levels better than mixture of Gaussians.

Object Identification

In this system the Zernike moments are used for detecting the object because they are noise immune and robust under partial occlusions compared to the other moment functions. They are less complex compared to SIFT and ASIFT method.

Tracking

The mean shift algorithm is used for real time implementation because of its robust statistical method which finds local maxima in the probability distribution.

Pose estimation

For pose estimation, we used Zernike moments because of highest accuracy compared to all other moments.

6.5. Results and Analysis

1. DSP/BIOS
Below shows some of the outputs generated using DSP/BIOS which are used for the performance analysis.

1.1 Execution Graph

The execution graph is a tool that shows the processing and states of SWI, PRD, TSK, SEM and CLK objects. The execution graph does not time stamp every event but keeps track of CLK and PRD events for time reference. Since it doesn't keep track of time stamps the time scale is not linear. Ticks corresponding to CLK and PRD events give time reference. An execution graph is shown in Figure 6.11.

![Execution Graph]

Figure 6.11: Execution graph

1.2 CPU Load Graph

CPU load is the amount of time the DSP is doing work compared to the total time or the number of instruction cycles used for work.
compared to total time. The time the CPU is not working is considered idle time.

Figure 6.12 shows CPU load graph. The peak CPU load is 99.67%.

Figure 6.12: CPU load graph

For the system at the base sample time is 200 ms and the CPU clock speed is 600 MHz, a much better solution to get the execution times is to use MATLAB’s built-in code profiler. Profile Report summarizes the maximum and average times for each subsystem.

```matlab
>> profile(CCS_Obj,'report');
```

Table 6.1 compares the execution times of our implementations on the PC (2.0 GHz Intel core 2 processor with 2GB of memory) and DM642 EVM board.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average Execution time per frame</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PC</td>
</tr>
<tr>
<td>Background Subtraction</td>
<td>355.91 ms</td>
</tr>
<tr>
<td>Detection</td>
<td>287.23 ms</td>
</tr>
<tr>
<td>Tracking</td>
<td>42.28 ms</td>
</tr>
<tr>
<td>Pose Estimation</td>
<td>327.11 ms</td>
</tr>
</tbody>
</table>

Table 6.1 Execution times comparison

Speedup = \( \frac{ST}{st} = 14.97 \)

Where \( ST = \) sum of execution times of PC and

\( st = \) sum of the execution times of DM642 EVM

Therefore, the average total execution time is 67.3 ms per frame. That yields 14 frames per sec which provide real time processing for the application.

Conclusion

The methods have been implemented on the Texas Instrument TMS320DM642 EVM board for real time processing. The software has been modified for embedded implementation. The on-board system can perform the line of procedures from frame grabbing to pose estimation as fast as 14 frames per second. That allows for a real time implementation of this complex tracking task.
CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this work, we have formulated a task of maintaining a coordinated flight by means of processing the optical data. We have developed a system able to recognize in real time an object of interest in on the background and track it for as small changes in as small rotation angles in 3D as 6 degrees and with the scale change between .3 and 1.4. For the development, we have studied a number of high efficiency algorithms. The study provides object background subtraction, detection, tracking and pose estimation of moving objects from video sequence.

For background subtraction task, the methods we evaluated are frame differencing, approximate median filter, mixture of Gaussians and proposed classification based on neural network methods. If the accuracy is concerned, the classification based on neural network performs better compared to rest of the algorithms. However when processing time is concerned the frame differencing is the fastest of all methods. A good trade-off is attained with the approximate median filter when implemented on DSP board.

For object detection, we have analyzed the performance of invariant moments, scale invariant feature transform and affine scale invariant feature transform methods. ASIFT and SIFT are more robust to noise and
scale changes and recognizes the object with 100% of accuracy even under 50% occlusion. Among the moment functions Zernike moments are noise immune, but degrade the performance with the scale increase and percentage of occlusion. Based on the complexity and accuracy constrains, we have selected Zernike moments for real time implementation.

Various tracking algorithms have been evaluated: mean shift with variable and fixed sized windows, the scale invariant feature transform, Harris and fast full search based on fast Fourier transform algorithms. Among all the methods SIFT features are scale invariant, more robust to illumination changes compared to other algorithms and quite stable to occlusions. However compared to mean shift and SSD FFT algorithms, SIFT is computationally intensive. Thus, for real time implementation we considered the mean shift algorithm based on the accuracy and time.

For pose estimation, a new method for finding the pose of the flying object based on moment functions approach is introduced. Zernike moment’s based look-up method appears to work accurate up to 6 degrees of rotation angle change.

Finally, the hardware solution is obtained with Texas Instrument TMS320DM642 EVM board yielding 14 frames per second.
7.2 Future Work

The future research is to be focused on the followings

- More accurate pose estimation using hybrid methods such as Combination of moments, SIFT, Harris etc.
- Use of stereo video for obtaining 3D features for accurate pose estimation.
- Fast methods of index tables and table look-up procedures.
- Expanding our pose estimation algorithm to multi-environments, like urban flight
- Speeding up the processing by porting operations to the graphics processing unit (GPU).
- If higher computational intensity algorithms show better accuracy, then a dedicated hardware can be thought and designed.
REFERENCES


20. Alan M. Mcivor, “Background Subtraction techniques”.


30. Mittal and D. Huttenlocher, “Scene modeling for wide area surveillance and image synthesis,” in Proceedings IEEE conference on


114. Texas Instruments, “TMS320C6000 DSP External Interface (EMIF), Literature Number: SPRU266E”, August 2004


118. Texas Instruments, “TMS320C6000 DSP TCP/IP Network Developer’s Kit”, August 2001

VITA

Graduate College
University of Nevada Las Vegas

Ajay Kumar Mandava

Degree:

• Bachelor of Technology, Electronics and Communication Engineering, 2006, Jawaharlal Nehru Technological University, India

Selected Publications:


Thesis Title: On-Board 3D Object Tracking: Software and Hardware Solutions

Thesis Examination Committee:

Chairperson, Dr. Emma E. Regentova, Ph.D.
Committee Member, Dr. Muthukumar Venkatesan, Ph.D.
Committee Member, Dr. Henry Selvaraj, Ph.D.
Graduate College Representative, Dr. Ajoy Kumar Datta, Ph.D.