2006

Machine vision applications in UAVs for autonomous aerial refueling and runway detection

Larry W. Rowe II
West Virginia University

Machine Vision Applications in UAVs for Autonomous Aerial Refueling and Runway Detection

Larry W. Rowe II

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Aerospace Engineering

Department of Mechanical and Aerospace Engineering

Morgantown, WV
2006

Keywords: Machine Vision, Marker Detection, Runway Detection

ABSTRACT

Machine Vision Applications in UAVs for Autonomous Aerial Refueling and Runway Detection

Larry W. Rowe II

This research focuses on the application of Machine Vision (MV) techniques and algorithms to the problems of Autonomous Aerial Refueling (AAR) and Runway Detection. In particular, real laboratory based hardware was used in a simulated environment to emulate real-life conditions for AAR. It was shown that the K-Means Clustering Algorithm solution to the Marker Detection problem could be executed at a frame rate of 30 Hz and it averaged a tracking error of less than one pixel while utilizing only 0.16% of the image. It was also shown that the solution to the Runway Detection problem could be executed at a frame rate of 20 Hz which is acceptable for use in an UAV performing reconnaissance work. Data from these tests suggest that both software schemes are suitable for applications in moving vehicles and that the accuracy of the measurements produced by the schemes make them suitable for UAV applications.

ACKNOWLEDGEMENTS

First, I would like to thank Dr. Marcello Napolitano for his knowledge, guidance, and support throughout my life as a graduate student. You will certainly not be forgotten.

I would also like to thank my colleagues Dr. Brad Seanor, Dr. Yu Gu, Dr. Giampiero Campa, Srikanth Gururajan, and Peter Cooke for the knowledge and guidance they gave me in completing this research. All seriousness aside, I would like to thank the group for all of the camaraderie, good times, and bad times we had in completing the Autonomous Formation Flight project. That is and will always be a memorable time in my life.

Special thanks to the NASA West Virginia Space Grant Consortium, Dr. Majid Jaraiedi, Dr. Mario Perhinschi, and Dr. Napolitano for the funding and the opportunity to do this research.

I would like to express my deepest appreciation to my wife, Shannon, for being an incredible supporting partner during this part of my life. At the same time I would like to express my deepest sympathy to her for all of the lonely nights she spent while I was working. I would also like to thank my son, Larry, for the inspiration he provided for me to finish this thesis.

Lastly, I would like to thank my parents, Larry and Connie, for their help in getting through college and their life long support in doing anything I ever wanted to do.

GO MOUNTAINEERS!

Table of Contents

List of Tables

List of Figures

Nomenclature

<u>Symbol</u>	<u>Description</u>
<u>English</u>	
x	X-component of a pixel in an image
y	Y-component of a pixel in an image
N	Number of rows in an image
M	Number of columns in an image
$f(x, y)$	Intensity of a pixel
<u>Greek</u>	
θ	Angle of normal line to the X axis
ρ	Length of normal line w.r.t. the origin
<u>Acronym</u>	
AAR	Autonomous Aerial Refueling
AVI	Audio Video Interlace
CCD	Charge Coupled Device
CMOS	Complementary Metal-Oxide-Semiconductor
COTS	Commercial Off The Shelf
CT	Computer Tomography
DDR2	Double Data Rate 2
DIP	Dual In-Line Package
DSP	Digital Signal Processing
DV	Digital Video
FLIR	Forward Looking Infrared Radar
FOV	Field Of View
FPS	Frames Per Second
FSB	Front Side Bus
GB	Gigabyte
GHz	Gigahertz
GPS	Global Positioning System
GUI	Graphical User Interface
Hz	Hertz
IMU	Inertial Measurement Unit
I/O	Input/Output
LED	Light Emitting Diode
MAV	Micro Air Vehicle
MB	Megabyte
MHz	Megahertz
MPEG	Moving Picture Experts Group
MRI	Magnetic Resonance Imaging
MSI	Micro-Star International
MV	Machine Vision
OBC	On-board Computer

PCI	Peripheral Component Interface
PIC	Programmable Integrated Circuit
PSD	Power Spectral Density
PWM	Pulse Width Modulation
RF	Radio Frequency
ROI	Region Of Interest
RGB	Red-Green-Blue
RMS	Root Mean Square
SATA	Serial Advanced Technology Attachment
SEAD	Suppression of Enemy Air Defenses
SDRAM	Synchronous Dynamic Random Access Memory
UAV	Unmanned Air Vehicle
US	United States
WVU	West Virginia University

Chapter 1

Introduction

1.1 What is Machine Vision?

Machine vision (MV) is the application of computer vision for several industry, manufacturing, and military purposes. While computer vision is primarily focused on machine-based image processing, MV encompasses and almost always requires digital input/output devices, computer networks, and/or software to control other equipment[1]. The term MV dates back to 1954 when Jerome Lemelson used computers to analyze digitized images from a video camera[2]. This was the beginning of several decades of research to develop various theories and computer algorithms to perform certain functions on images. Theoretical advances that form the basis of modern MV are now more than 20 years old and one needs only to review the contents of *A. Rosenfield and A. Kak, Digital Picture Processing, Academic Press, vol. 1-2, 1982* to confirm this assertion[3]. Thus, a MV system is simply a computer-based system that is capable of capturing or grabbing an image and performing some sort of analysis on it. So, this would lead one to believe that the recent history of MV is essentially the adaptation of evolving computer technology to the commercialization of image processing for automation[3].

In the 1970s, mainframe computers were first coupled with image capture devices and external display peripherals. With the advent of the desktop PC in the 1980s, having a computer dedicated to MV was no longer so difficult and special purpose hardware designed to accelerate image processing was readily available. During the 1990s, MV followed trends set by current computing platforms such as Windows® and Linux®. High performance microprocessors were available at a fraction of the previous cost and the ability to perform many tasks previously performed by digital signal processors (DSP) was now shifted primarily to software and this made special purpose DSP MV hardware virtually obsolete.

In recent years, MV has evolved into a highly integrated field involving many disciplines of engineering such as computer science, optics, mechanical and/or aerospace engineering, and automation. This further enhances the diversity of the applications of MV to include a number of engineering topics. Within these topics, MV is used to help solve problems or perform tasks which, otherwise, would be too expensive, unreliable, or dangerous for human involvement.

1.2 Problem Definition

This research effort is divided into two distinct phases. The first phase deals with the detection and tracking of multiple markers or light markers attributed to a tanker aircraft in the field of view (FOV) and is referred to as the *Marker Detection and Tracking* phase. The purpose of this phase of the research effort is to address the Marker Detection and Tracking problem, and to develop and evaluate hardware and the appropriate software tools and approaches. This includes the research, acquisition, and evaluation of commercial off the shelf (COTS) hardware that can facilitate MV laboratory experiments. The problem further involves the development and evaluation of software that can detect light sources or markers on an aircraft using live video and track the object as it moves in the FOV. This phase of this research effort was funded in part by Dr. Majid Jaraiedi and the NASA West Virginia Space Grant Consortium.

The second phase of this research effort deals with the detection of a road, pipeline, or, in this case, a runway, thus, referred to as the *Runway Detection* phase. The purpose of developing a MV Runway Detection algorithm was to investigate the feasibility of such a solution and determine its real-time applicability to unmanned aerial vehicle (UAV) technology. The goal of this research is to develop software that can detect a runway, road, or pipeline in a video stream. The research effort focused mainly on software and on some hardware related issues and items. Specifically, the research performed earlier on laboratory MV hardware was not necessary for this software development.

1.2.1 Marker Detection and Tracking

Marker Detection and Tracking is only a small part of a much larger set of problems leading to the mutual goal of Autonomous Aerial Refueling (AAR). Initially, the Marker Detection and Tracking problem became a topic for discussion, due to the need to examine a number light emitting markers attached to an aircraft in various places through MV. This was coupled with needs defined by a current research project focusing on AAR at West Virginia University (WVU). The idea is that if the markers could be reliably detected and tracked by an UAV, then the position information of the markers could be used to estimate the position of the UAV relative to the tanker aircraft. This problem is known in the technical literature as the pose estimation problem. This information would then be used to drive a control system whose goal would be to guide the UAV to the refueling position behind the tanker.

Several methods have been researched for performing guidance for the AAR problem. The first method that was researched was the 'GPS Only' method. This method used the global positioning system (GPS) to attempt to guide the UAV into refueling position with the tanker. Several problems arose from this method. One problem was that the accuracy of GPS was not high enough to enable to UAV to dock with the tanker. The second problem was that the tanker would sometimes block the UAV's view of the satellites above, thus causing it to lose even more accuracy by using a reduced number of satellites for a position fix. The second method that was researched was the 'MV Only' method. This method used machine vision only to guide the UAV into refueling position. This method did not work well at large distances due to the size that the tanker would appear to the UAV's camera. The third method that was researched was the 'GPS + MV Sensor Fusion' method. This method uses GPS for large distances where the MV does not work well. It then uses a combination of GPS and MV for the intermediate distances. Finally, it uses MV only at close distances to complete the docking process. This is the methodology that has continued to be researched today and it has been shown to work very well. An illustration of the 'GPS + MV Sensor Fusion' method can be seen on the following page in Figure 1.1.

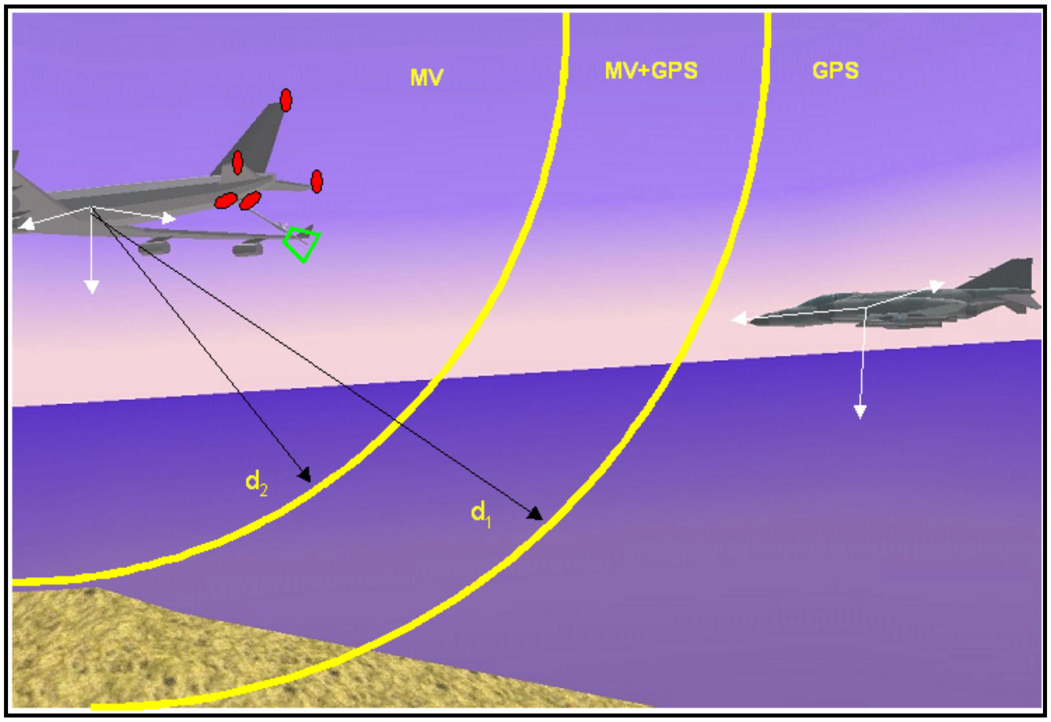


Figure 1.1: 'GPS+MV' Approach with a Single Set of Optical Markers

This problem could be called Marker Detection only, but the goal was not limited exclusively to finding these markers. Finding the markers in the fastest possible manner was key to the operation due to the desired real-time applicability. Tracking the markers once they were located was found to be a much faster method for performing this task. Therefore, this is where tracking the marker plays an important role. Originally, the markers were found by constantly scanning the current image for them. It was then thought that if the markers could be found by scanning the entire image only three times initially, that some inertial information about the movement of the markers could be derived and the next position could be estimated. From this estimated position, the search area could be reduced from the entire image area to a small area around the estimated position, thus tracking the marker locations and enhancing the speed of operation.

As with any problem, there are certain design constraints that should be addressed. For this problem, there are three factors that had to be considered. First, a decision must be made during the design phase to determine if the software should be made real-time capable or to have it executed as an external process on a ground-based computer. The reasoning behind the necessity for this decision is that if the software could not be made in a compact enough form to allow its use in an on-board computer (OBC) in a UAV, then it could be executed on a ground based computer and the input/output information relayed via radio frequency (RF) transmissions. The second decision deals with the issue of the number of markers there are to find. Obviously, if the number of markers increase, the computer workload will also increase and the computation time will be increased and vice-versa. The pose estimation algorithm may also have constraints with respect to the minimum number of markers required to obtain an accurate pose estimate. Lastly and probably the most important thing to consider is the physical constraints that should be in effect to ensure robust operation of the software. This depends mostly on the initial conditions of the software and the attitude of the aircraft in question when the software is executed and there are certain situations that should be avoided which will be discussed in Chapter 4.

Although these decisions present themselves to the designer fairly obviously, there are some other things that have to be considered such as what exact mathematical approach should be taken to solve this problem. Up until recent years, a considerable amount of research has been directed towards developing methodologies to cut through the medium level processing to reach a point where it is sufficient to extract features from an image based solely on a thresholding process. The result of this is that the complexities of some cluster detecting algorithms have been reduced, making them more attractive. Despite these efforts, most current cluster detecting algorithms can be characterized as a fairly unorganized collection of concepts. Since researchers have been tailoring the algorithms to their specific applications, there has been no consensus on a generic cluster-detecting algorithm.

While tailoring algorithms for a specific application is practical and has produced many useful problem-solving hints, the lack of a general concept on cluster detection has made the interpretation and extension of the algorithms difficult. As a result of this, most cluster detection problems still remain to be done manually and it is more efficient on many levels to perform brute force detection and tracking for each individual purpose as it presents itself.

1.2.2 Runway Detection

The problem of road, pipeline, or, in this case, runway detection is a topic which has spurred much interest since heightened security in the United States (US) and around the world has been a major concern. The availability of an UAV that could autonomously fly above these critical parts of infrastructure for the purpose of monitoring their condition could be very useful to many security agencies such as the U.S. Department of Homeland Security and the U.S. Border Patrol. This type of technology could be used in UAVs to allow them to be used to monitor these points of infrastructure without spending massive amounts of money required to operate a manned reconnaissance type aircraft. Therefore, this phase of the research effort was inspired by the this interest in this subject.

In essence, this is a problem of image segmentation. The image can be segmented into two sections: that is the runway and everything else. In order to do this, one must look at what features can be extracted from the runway image. The most striking feature of a runway is the straight lines. Based on this defining feature of runways, line-detecting algorithms and methods will be researched for use in solving this problem. The aim of this research effort combines an actual hardware setup and a software based method with the ability to test algorithms that could reliably detect a runway in a video sequence in a near real-time manner. The trajectory of the straight lines could further be used as an input to a guidance system in a UAV to allow it to follow the straight-line object of interest. Based on the desire to use these methods on an UAV, the pre-processing stage is one area that is examined closely since the pre-processing sequence commonly demands more time with a frame than the main processing algorithm does. In an effort to make the computation time and computer workload as small as possible for use in an UAV, it is vital to the operation that the pre-processing sequence be as efficient as possible. Therefore, it is important to find the correct sequence of filters to yield the smallest processing time, yet, still yield acceptable results.

Before a line-detecting algorithm can be decided upon, however, the design constraints should once again be addressed. For this particular problem, the design constraints are difficult to narrow down, but two things immediately stand out as potential problems in this research. The first involves the desire to make this algorithm perform in real-time or in near real-time fashion. This fact greatly influences the decision of the software environment, the algorithms used, and the complexity of the filtering process in order to achieve the goal. Often, it is not the main algorithm that uses most of the processing time; the pre-processing stage might be main bottleneck instead. The second problem is also clear and that is that there may be other things in the image that have prominent straight lines. The immediate things that come to mind are roads, rivers, and the horizon. These are all things that could skew the results while using any candidate line-detection algorithm. Therefore, it seems that equal emphasis should be placed on the algorithm, as well as the pre-processing sequence.

The pre-processing sequence must be responsible for filtering out these other straight line ‘artifacts’ that are not the runway. It has to be able to present the line-detecting algorithm with an image that is free of anomalies and free of artifacts that could be mistaken for a runway. As a result of this, the pre-processing sequence has the most important function in the entire scheme in that it must be able to execute in a timely fashion and it has to assuredly filter out all unwanted things from the image. The importance of the pre-processing task can be reduced slightly, however, by having a line-detecting algorithm that is robust to runway imposters. In order to achieve this, an error correcting feedback loop will be needed.

1.3 Research Objectives

The following research objectives are intended to address the development and evaluation of MV hardware, as well as to develop and apply MV algorithms to the problems of Marker Detection and Tracking and Runway Detection.

Task #1. Select and integrate a set of MV hardware capable of testing MV algorithms in a lab environment.

Task #2. Develop and test MV software using the Matlab® programming environment, which can detect light sources on a tanker style aircraft. This software development shall coincide with ongoing AAR research at WVU. This task involves the development of different approaches to the problem of the detection of light sources. Possible experiments used to validate this task include aircraft roll angle measurement and validation, repeatability analysis, and timing profiles.

Task #3. Develop and test MV software using the Matlab® programming environment, which can detect and track light sources on a tanker style aircraft. Possible experiments used to validate this task include aircraft roll angle measurement and validation, repeatability analysis, and timing profiles.

Task #4. Compare the results of Task #2 and Task #3 in the accuracy of the aircraft roll angle measurement, the statistic profiles of the repeatability analysis, and the timing profiles.

Task #5. Develop a MV software scheme using the Simulink® programming environment, capable of detecting things such as roads, runways, and pipelines. This software development coincides with current interest shown by security agencies in using UAVs to monitor critical parts of infrastructure..

Task #6. Evaluate MV software scheme in Task #5. This is accomplished by utilizing videos acquired via a hardware platform to be developed that facilitates video acquisition from an existing WVU UAV. Possible experiments used to analyze the performance of this scheme include comparing calculated attitude parameters to parameter data recorded by an OBC on a WVU UAV.

1.4 Overview of Thesis

This thesis is organized as follows. Chapter 2 presents work by other researchers in the area of image segmentation methods ranging from the most simple to the more complex, as well as their application to the aerospace industry.

In Chapter 3, the theory behind the pre-processing algorithms and the main solution to each problem is presented and discussed. In particular, the algorithms used to find and track the markers and the line-detecting algorithms are covered here.

Chapter 4 is dedicated to the experimental setup of the solution arrived at by this research effort. This includes the description of the pre-processing steps used in each problem as well as the description of the implementation of the main algorithms in software. The setup of the hardware required for obtaining results from laboratory experiments is also covered here.

Chapter 5 presents the results obtained from laboratory experiments involving both the Marker Detection and Tracking algorithm and the Runway Detection scheme. These results include comparisons in performance and robustness as well as statistical information regarding repeatability of results.

Chapter 6 contains the conclusions drawn from this research on these problems and also the recommendations for future work involving the research presented here.

Chapter 2

Literature Review

2.1 General

MV has traditionally been applied to industrial or manufacturing settings because of the size and weight of the equipment required. But, in the last decade, MV has gained a promising outlook as to its feasibility of use in aerospace applications requiring real-time solutions. This new outlook for MV is not a realization of new MV techniques or theory, but a realization of the advancement of semiconductor technology resulting in the ability to manufacture faster, lighter, more efficient machines which can handle the heavy loads of MV applications.

In general, the next step for MV technology is of course going to involve smaller, faster, and more efficient technology. This technology will most likely focus on total integration of the entire MV system into a single sensor. This *vision sensor* would be required to be network ready and contain integrated DSP to ensure at minimum, performance that matches current MV systems. It would also require the ease-of-use of the current generation of MV systems, but at a lower cost. Distributed computing techniques will most likely be involved, making the *vision sensor* a self-sufficient network resource. Point-to-point dedicated user interfaces would become obsolete and vision sensors would be able to cooperate in peer-to-peer groups; able to perform multi-camera, multi-angle processes which currently are very bulky to consider. In the coming years, expect to see the clear emergence of the *vision sensor* paradigm in, what is conceivably, the ultimate step in the evolution of conventional MV hardware[3].

2.2 Image Segmentation Methods

Image segmentation has a very broad research base, which varies greatly depending on the application. Applications of MV have been researched in many fields including medical, biology, agriculture, and aerospace engineering. Most of the research in image segmentation has been done in the medical field. With respect to the aerospace field, image segmentation has been used for quite a while but in very specific applications with automated rendezvous and docking (AR&D) being the main topic. In the last decade, however, image segmentation has become a more prominent topic in the aerospace industry. With the realization of new technology that enables computers to be placed into smaller and lighter packages while maintaining the speed and reliability seen in the past, image segmentation and MV in general has been applied to many more things than it could have years ago. This technology has allowed a more complex MV system to be incorporated into a lighter and smaller area allowing it to be used in many applications where space is at a premium and this is especially true with the major push in the field of UAVs that is being seen today. Since there is more interest in image segmentation due to the availability of technology, this has caused the research base to evolve to include many more techniques than would have previously been addressed when talking about image segmentation.

The main topic, segmentation, can be defined as distinguishing objects from the background. For intensity images, which are those images being represented by point-wise intensity levels, the four popular approaches are: pixel-based methods, edge-based methods, region-based methods, and connectivity-preserving relaxation methods[4]. These methods will be described in detail in Section 2.2.1 through Section 2.2.4, respectively.

2.2.1 Pixel Based Methods

Threshold techniques, which make decisions based on local pixel information, are effective when the intensity levels of the objects fall squarely outside the range of levels in the background[4]. Therefore, any image that contains objects that have a blurred boundary with respect to the background, will be difficult to detect with this method. This downfall makes this technique difficult to apply reliably by itself, but it is possible that it would be much more effective when applied in conjunction with a more advanced segmentation method. Pixel based methods of image segmentation can be further broken down into two parts. The first part mainly deals with a low-level segmentation method called thresholding. The second part deals with a few of the more advanced pixel-based segmentation methods. Both aspects of pixel-based methods will be fully explored below.

Image segmentation performed by thresholding is the simplest form of segmentation. Because of this, there are a wide number of variations on the use of thresholding for image segmentation purposes and many of them are only precursors to a more advanced segmentation method. An example of thresholding being used as a stepping-stone to a more advanced segmentation technique can be found in recent research performed by Deshmukh and Shinde[5]. This research investigates the possible methods that could be used to perform color-based image segmentation such as region growing, neural network based, and fuzzy based techniques. In each of the methods, thresholding is either used as a low-level technique or as the main technique acted on by the adaptive nature of neural networks or fuzzy logic.

Clustering falls in the group of more advanced pixel based methods and is defined as the process for grouping data points with similar feature vectors together in a single cluster[6]. A feature vector may consist of the gray values, contrast values, and local texture values or measurements for each pixel in the image. This type of clustering frequently produces disjoint regions where there may be holes or disconnections in regions that are supposed to be connected. Therefore, post processing of some type that will allow the disjoint regions to reconnect as one region is usually necessary.

There is one main clustering algorithm with two variations used throughout image processing. These three variations are all based on the K-Means Clustering algorithm. The use of the K-Means algorithm alone is the most common, accounting for approximately 70% of the use in clustering problems. The other two variations account for the other 30% of use and they are the FUZZY C-Means Clustering Algorithm and the Adaptive FUZZY C-Means Algorithm. These two methods are slightly more advanced than the stand alone K-Means algorithm due to their adaptive nature. Most industry use of K-Means occurs with the stand-alone algorithm with the two variations currently being used only in the high-level research environment. These two variations are fairly new concepts and as such, they have not been applied to industry in any significant numbers. As a result, the focus of the review of clustering methods will be on the stand alone K-Means Clustering Algorithm.

The major drawback to the K-Means Clustering Algorithm is that a priori knowledge of the number of clusters is needed to accurately make the algorithm work. Many researchers are addressing these issues by using a hybrid, spectral clustering[7,8], neural networks[5,9], a hybrid of the stand-alone K-Means[10], and a hybrid of the Adaptive FUZZY C-Means Algorithm[5,11,12] mentioned earlier. A few more places where the K-Means Clustering Algorithm can be found is in vision systems used by robots[13] and in the IT sector, where researchers have tried to improve the speed of image search engines by clustering similar images[14]. These experimental methods are sure to move to the forefront of technology when they are perfected enough to be reliable when used in an everyday environment.

2.2.2 Edge Based Methods

Edge-based methods center around contour detection and their weakness is also a blurred boundary. This causes a weakness in their ability to connect together broken segments of a single contour line. This, in turn, will cause the software to detect several contours instead of a single one. Ultimately, this weakness propagates into increased computational workload because each contour must now be assessed rather than dealing with one big contour line. Like thresholding, these methods are also likely to be much more reliable when used in conjunction with a more advanced segmentation method. Edge-based segmentations rely on edges found in an image by edge detecting operators – these edges mark image locations of discontinuities in grey level, color, context, and etc[15]. There are many different edge-detecting operators such as Sobel, Canny and Roberts, but the image resulting from the use of these operators cannot be used as a segmentation result. Other processing steps must follow to combine edges into contours that correspond better with borders in the image[15]. Discussion will follow for the two main methods of edge-based segmentation and a common higher-level method.

In an edge image, small edge values correspond to insignificant grey level changes resulting from quantization noise or small lighting irregularities[15]. Sometimes, thresholding of an edge image can be used to remove the small edge values. Thresholding an edge image simply filters out the more faint edges or noise, whatever they may be. If the original image has high contrast, this method will work, but if the image is noisy, this will result in errors. Graph searching is another method of edge-based segmentation. The simplest, and also the least effective method of grouping edges is to use heuristic search[15]. This means the algorithm would start on a boundary pixel and try to join neighboring pixels based on their edge strength and direction. After this is complete, some thinning such as the use of a skeleton algorithm would have to be used to remove pixels at places where the edge line is more than one pixel thick. Also, Brejl and Sonka present an automated model based image segmentation algorithm whose basis is the edge-based segmentation method[16]. By adding additional algorithms to automate the edge detection process, the two major edge based segmentation problems mentioned previously are addressed.

These edge-based segmentation algorithms are very effective when used with clean images. But, the most common problems of edge-based segmentation, caused by image noise or unsuitable information in an image, are an edge presence in locations where there is no border, and no edge presence where a real border exists[17]. Hence, they can suffer from inadequate sensitivity and specificity because the image in the gradient space must be thresholded or otherwise classified according to edge or non-edge membership[18]. Also, the problem of tracking an edge that bifurcates into two or more edges is one that cannot be adequately resolved using these low-level image operators alone[18].

There exists a considerably more complex edge-based segmentation method known as the Hough transform. The Hough Transform was named after Paul Hough who patented it in 1962 as a highly effective method of utilizing mathematics to describe boundary curves in images[19]. The original Hough transform was designed to detect straight lines and curves and this original method can be used if analytic equations of object borderlines are known -- no prior knowledge of region position is necessary[20]. This is an extremely desirable trait of segmentation algorithms because it allows much more flexibility in initial conditions or changing conditions. The greatest advantage of this method is the robustness of the segmentation results; that is, segmentation is not too sensitive to imperfect data or noise[20]. Since the Hough Transform has been around almost as long as the term MV has (1954), one would expect there to be many sources for information and there are. This review of the Hough transform focuses on the most recent uses in industry and in research.

2.2.3 Region Based Methods

A region-based technique can be considered to be a more advanced segmentation method. A region-based method usually proceeds by partitioning the image into connected regions by grouping neighboring pixels of similar intensity levels. Adjacent regions are then merged under some criterion involving perhaps homogeneity, sharpness, or region boundaries. The downfalls of this method are that over-stringent criteria can cause fragmentation and criteria that are too lenient will overlook object boundaries and can cause many objects to be grouped as one.

Region-based image segmentation is a technique whose purpose is to separate the image into meaningful, non-overlapping regions, which would be used for further analysis[21]. Since the 60’s, a variety of techniques have been proposed for segmenting images by identifying regions of some common property[22]. These can be classified into two main classes. The first is merging algorithms in which neighboring regions are compared and merged if they are close enough in some property[22]. The second is splitting algorithms in which large non-uniform regions are broken up into smaller areas, which may be uniform[22].

Merging must start from a uniform seed region. One method of determining a suitable seed region is to divide the image into 4 or 16 pieces and check each one for similarities. Another approach is to divide the image into strips, horizontally or vertically, and check each strip against each other for similarities. The worst case would be when the seed is a single pixel. Once a seed is found, each similar neighboring region is merged until no more similar regions can be found. As one might imagine, there is a major drawback to this method. This process is inherently sequential, and if fine detail is required in the segmentation, then the computing time will be long[22].

The splitting algorithms begin from the whole image and divide it up until each sub region is uniform. The usual criterion for stopping the splitting process is when the properties of a newly split pair do not differ from those of their original region by more than a threshold[22].

Given the explanation of these two main methods, one can immediately assess the problems that would be encountered in trying to apply these methods in a real-time situation. Computation time, human interaction to select the seed, and uncertain results all come into play when assessing the feasibility of using these methods. Kothe[23] has evaluated the use of these methods in a post-processing manner. In this environment, these methods work fairly well, except that they require some smoothing operations, which always remove some details. It is evidenced in this paper that the computation time and the reduction of detail make these algorithms usable only in a one-time use type of way. It is evident that with all of these problems, these methods can be overlooked as a feasible solution to their use in this research.

2.2.4 Connectivity Preserving Relaxation Methods

The connectivity preserving relaxation based segmentation method, usually referred to as the *active contour model*, was proposed recently[4]. This method starts with some initial boundary shape that is represented by splines and iterative modifications are made to that shape using various shrink/expansion operations according to an energy minimizing cost function[4]. Given the inherent complexity of splines and the added complexity of a constantly evolving set of them, this method can most definitely be categorized as computational intensive. Therefore, due to the required computational effort, this method would probably not be a feasible solution in a real-time environment. Since this method has just been recently proposed and is computational intensive, this is the least researched method of the four methods discussed. Given these circumstances, this method is probably not applicable to the aim of the research described here and will not be discussed further.

2.3 Image Segmentation Applications

Applications of image segmentation and especially the four methods previously described have been researched extensively. The information gathered from the research on their application to scientific problems is presented below in Sections 2.3.1 through Section 2.3.6. These sections have been broken down into their respective scientific areas and the applications of all of the methods to these areas are included.

2.3.1 Aerospace Related Image Segmentation Applications

As previously mentioned, MV has a promising outlook for applications within the aerospace industry. Currently, many government agencies and universities are performing research involving MV. Most research involves the replacement of a human with MV technology to eliminate having to put a human in harm's way. Research in the aerospace industry has began to involve UAVs, which inherently do not carry humans. Therefore, to extend the capabilities of an UAV to approximate that of a manned aircraft, MV is one possible solution that is being investigated.

The trend of increasing use of UAVs in order to eliminate the human risk factor involved in the Suppression of Enemy Air Defenses (SEAD), general reconissance, and/or high risk, high value missions will certainly continue. These UAVs are very attractive in that they eliminate risk to the human crew while performing these dangerous missions, the aircraft have potential for greater survivability, they have greater endurance to perform a mission as opposed to crew fatigue, the cooperative nature gives a greater probability of successful outcome, and finally cost is reduced[24]. Given this information, the ability for a UAV to detect objects on the ground and in the air will be vital to their functionality and survivability. The ability to detect threats on the ground or to be able to refuel itself to endure longer flight times are major objectives that can be met by the use of MV. In this review, a many applications of image segmentation were found that directly relate to research in the aerospace industry. These applications will be highlighted below.

AAR has been an extensively researched topic for the last several years. Many universities such as Texas A&M[25,26] and WVU[27,28,29,30,31] as well as the United Stated Air Force (USAF)[32] has ongoing research in this area. The most recent research effort at WVU involves semi-AAR in a real-time system[28] and using feature extraction[29] and corner detection[31] to determine the pose of the tanker with respect to an UAV. Research in this area has focused on enabling an UAV to refuel without human intervention. Previous research[27,30] has taken many paths including active marker based vision where the tanker would have light emitting 'markers' placed in an array on the underbelly and tips of its empennage. The idea is that the UAV would then be able to sense the 'markers' and by the use of labeling techniques and feature matching algorithms the markers would be labeled as to their actual location on the tanker. Then, pose estimation such as the Gaussian Least Squares Differential Correction[27] (GLSDC) or the Lu, Hager and Mjolsness[27] (LHM) algorithm would enable to UAV to determine its 'pose' with respect to the tanker and the UAV would then orient itself correctly with the tanker using a control system and move into refueling position.

This research has now been focused in another direction in which the tanker would have no light emitting markers in the visible spectrum due to the risk presented in revealing an aircraft's location to an enemy at night. The UAV would then have to discern its pose information from other methods using MV, namely feature extraction[29]. Research at Texas A&M has focused on using a vision based navigation sensor[26] for AAR purposes and developing a robust trajectory tracking controller for the probe and drogue type of refueling apparatus[25].

There has also been extensive research in the area of autonomous formation flight by researchers at WVU[33,34] and Georgia Institute of Technology[35,36]. The need to find ways of maintaining robustness in a formation flight system is important due to any number of circumstances. The number one thing that can affect formation flight is communications. If communications are lost, the inability for a leader aircraft to send its position to the follower aircraft will cause the follower aircraft to do very undesirable things. So, in an effort to improve the robustness of a formation flight system, the addition of a MV system has been investigated by a joint West Virginia University and University of Pisa team[33]. In this research, five lighted markers were placed on a simulated leader aircraft. The follower aircraft was able to use its vision system to 'see' the markers on the leader. After finding the markers, pose estimation algorithms were used to estimate the position of the follower relative to the leader and then control algorithms were able to control the follower to accurately follow the leader.

Another example of vision-based control is shown in research in a collaborative effort between the University of California at Berkeley and the University of Colorado at Boulder. Researchers there have used the Hough Transform to perform a very complex job for an autonomous aircraft[37]. This job involves the autonomous following of a road using a small aircraft. Using the Hough Transform combined with other pre-processing techniques, the research team at the AINS Center for Collaborative Control of Unmanned Vehicles have been able to build and flight test their small UAV which includes a MV system. Their UAV was able to follow a road for over two miles before they had to end their test due to hardware constraints[37]. Further research has been performed in comparing various lateral controllers used in performing this function. Flight tests have not been conducted but simulations have been performed under ideal conditions comparing several aim-ahead controllers, sliding surface controllers, linear quadratic Gaussian (LQG) regulators, and a receding horizon controller (RHC)[38,39]. This is a perfect example of the uses of the Hough Transform that will be seen in the future in the aerospace industry.

Stability and control using vision systems is also a widely researched subject. Perhaps the easiest use of a vision system for stability and control is the ability to detect roll angle. By detecting the horizon, the roll angle is easily found. This has been applied both in simulation by researchers at Monash University in Australia[40] and researchers at the University of Colorado at Boulder[38] and in flight testing of micro UAVs by researchers at the University of Florida[41]. In both instances, the horizon was detected using either the Hough transform[40,41] or the Adaptive Receding Horizon[38] method and then the roll angle was estimated by finding the relative angle of the line detected to the artificial horizon defined by the camera orientation.

The researchers at Monash University[40] achieved their goal by using a robotic arm to rotate an artificial horizon image. The main purpose of this research was to do this task with very few components, for very low cost, and at low computational cost. This was achieved by using a programmable integrated circuit (PIC) microcontroller and not a standard computer like has been used in so many UAVs. The research at the University of Florida[41] was performed on micro air vehicles (MAV) built at the university. Since the MAVs were so small, a unique vision system had to be created. This was accomplished by using a type of embedded processor similar to the Motorola MPC565. By using this type of processor, the bulky computer parts normally seen in a UAV hardware suite was eliminated while still being able to perform the desired tasks. In both cases, the horizon detection problem was very well addressed and the results were impressive.

Researchers at Drexel University in Philadelphia have tried a different approach. By using a blimp as a UAV they have been able to extend flight times almost indefinitely and by doing so, can accomplish much more research per flight than can be accomplished with a conventional UAV[42]. In reference to their research topic, this involves collision avoidance and following a simulated road with a payload that weighs less than 100 grams. The collision avoidance is accomplished by using an optic flow sensor. An optic flow sensor is not a camera, it is a sensor that will output a higher voltage if it ‘sees’ a lot of things. For example, if the optic flow sensor were to be placed in the middle of a room, the output would be fairly low. But, if the sensor were placed in front of a bookshelf, a wall, or a person, the output would be fairly high. So, by monitoring the voltage coming from this sensor, the blimp can determine if it is getting close to something and begin to reverse its motors to stop or back up in order to avoid a collision. The vision system is based on a small wireless camera that weighs 15 grams. This camera transmits its images down to a ground based vision computer, which analyzes the image and then calculates flight control commands based on a proportional-derivative controller. The commands are then translated into pulse width modulation (PWM) and sent to the receiver in the blimp. This very simple vision system is able to perform lots of things just by changing the software on the ground-based computer. Current research focused on being able to follow an artificial road, which was set up in an auditorium.

Another example of MV used for obstacle avoidance is with research preformed in a collaborative effort between the University of Missouri, Texas Tech. University, and the USAF[43]. This research focuses on various ideas and approached to deal with image noise in motion analysis. This research, like other research in robotics uses a range map to define the distance to objects with the field of view (FOV). This range map is then used for collision avoidance along with control and guidance laws designed to navigate the UAV between waypoints and avoid obstacles. This is a prime example of the direction of future research in coupling the diversity of MV with the advanced problems in the aerospace industry.

One current production MV application currently in use by military and other government agencies around the world is the forward-looking infrared radar (FLIR). The FLIR has had many variations in its lifetime, but the current FLIR used by US government armed forces and agencies as well as dozens of international governments and organizations is the FLIR Star SAFIRE™ HD[44]. This FLIR radar ball employs MV technology that can track, range find, and laser illuminate targets at extremely high resolutions at up to a 25 kilometer range in the Near, Mid, and Far Infrared and Visible light frequencies. All of these features come in a package that is less than 100 pounds, which makes this an incredibly viable package for any aircraft, but UAVs in particular. This type of MV technology is on the forefront and will only continue to improve as technology allows. The FLIR Star SAFIRE™ HD can be seen in Figure 2.1.

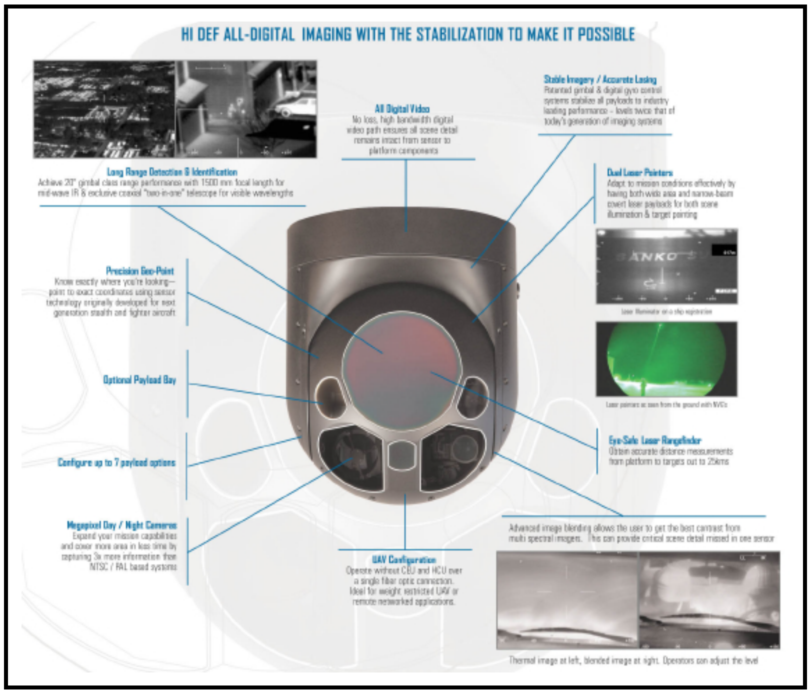

HI DEF ALL-DIGITAL IMAGING WITH THE STABILIZATION TO MAKE IT POSSIBLE
Long Range Detection & Identification
Achieve 20" gimbal class range performance with 1500 mm focal length for mid-wave IR & exclusive coaxial "two-in-one" telescope for visible wavelengths
All Digital Video
No loss, high bandwidth digital video path ensures all scene detail remains intact from sensor to platform components
Stable Imagery / Accurate Lasing
Patented gimbal & digital gyro control systems stabilize all payloads to industry leading performance – levels twice that of today's generation of imaging systems
Dual Laser Pointers
Adapt to mission conditions effectively by having both wide area and narrow-beam covert laser payloads for both scene illumination & target painting
Laser Illuminator on a ship registration
Laser pointers as seen from the ground with NVG's
Precision Geo-Point
Know exactly where you're looking—point to exact coordinates using sensor technology originally developed for next generation stealth and fighter aircraft
Optional Payload Bay
Configure up to 7 payload options
Eye-Safe Laser Rangefinder
Obtain accurate distance measurements from platform to targets out to 25kms
Megapixel Day / Night Cameras
Expand your mission capabilities and cover more area in less time by capturing 3x more information than NTSC / PAL based systems
UAV Configuration
Operate without CEU and HCU over a single fiber optic connection. Ideal for weight restricted UAV or remote networked applications.
Advanced image blending allows the user to get the best contrast from multi spectral imagers. This can provide critical scene detail missed in one sensor
Thermal image at left, blended image at right. Operators can adjust the level

Figure 2.1: FLIR Star SAFIRE^{TM} HD[44]

Since thresholding based segmentation is the most widely used method, it is expected that the use of this technique can be found in many places around the world, including space. This simple technique is being used to detect human settlements in images acquired by the IKONOS satellite[45]. Since the imagery from IKONOS is in 4-meter resolution, one can expect that there are almost an unlimited number of images of the earth to be processed. In order for scientists and researchers to process all of this information quickly, low-level image processing is required. By using the multi-spectral imagery from IKONOS, separating the regions of farmland from regions of housing is a pretty simple task easily achieved by thresholding. As evidenced from previous examples, thresholding is mostly a low level technique that is mainly used in conjunction with other segmentation techniques. Rarely, is the use of thresholding enough to complete the task at hand in an image analysis problem. Although, when thresholding is enough, it is a very fast and simple approach. Occasionally, thresholding can be applied to a complex problem such as IKONOS with great reliability, accuracy, and speed, which was essential to the objectives required with IKONOS.

2.3.2 Other Engineering

Aside from the field of aerospace engineering, uses of MV can be found in other areas of engineering such as civil engineering and electrical engineering. Several examples can be found of researchers in these areas using edge-based segmentation to perform some sort of image analysis. For example, edge based segmentation is used in mapping rock fractures[46]. According to the researchers, rock fracture mapping is an important task in rock engineering and making the algorithm robust is the hardest part[46]. According to Wang, using a valley-edge based segmentation algorithm is the first step in creating a robust algorithm.

Some research has been performed to stress and highlight the robustness to noise of the Hough transform.. Range images, which are images that are used to judge distances, are subject to noise due to weather, lighting, and stray objects that may be in the field of view. Robots acquire range images and process them to determine distances to various objects in the field of view so that they may calculate how long to power their motors to travel to the object[47]. In doing this, range images must be evaluated quickly and accurately and the Hough transform can do just this. This method of analyzing range images is described by Gatcher of the Ecole Polytechnique Federale de Lausanne and his research shows that when compared to various other image processing techniques used to do this same job, the Hough Transform is more accurate and faster than any other method[47].

The robustness of the Hough Transform has, again, been proven by researchers at the University of Puerto Rico. They have shown that the major advantage of using this transform instead of any other techniques is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise[48]. This extreme robustness makes the Hough Transform an ideal method of line extraction and image segmentation in high-risk applications where it is essential that no mistake be made about the results of the analysis being performed.

2.3.3 Manufacturing Industry

Aside from the field of engineering, MV is also used in other areas of everyday life, especially in the manufacturing industry. With the current methodology and technology, MV systems are generally limited to performing narrowly defined tasks such as inspecting food products on a conveyor belt[49], tracking lift trucks in an industrial setting[50], or inspecting semiconductor chips[51]. The manufacturing industry favors MV systems because they can provide continuous, repeatable, high speed, high magnification inspections. Humans have traditionally catered to these tasks, but it is widely known that humans are often affected by distraction, illness, and boredom, which can jeopardize their perception over long periods of time. Although adapting MV systems to new quality control policies and outlying defects can be time consuming and problematic, MV systems provide a clear solution to the manufacturing industry to alleviate the economic effects of missed defects and costs associated with having to employ human inspectors.

Another current use of MV is in the food industry[49]. Camera based inspection systems are commonplace in just about every manufacturing plant for edibles around the world and has been for many years In order to ensure the expiration dates and lot codes are properly printed on many perishables, food manufacturers use MV systems. These systems are much faster than humans and are more accurate; they also ensure almost 100% trouble free operation for the fast moving production line. A typical food inspection system is shown in Figure 2.2. This system uses a camera to capture an image at the correct time when the container passes on the conveyer belt. The computer then uses character recognition software to analyze the image and make sure the correct characters are present on the bottom of the container. The computer will then make a decision to either let the container go to the next stage of production or to remove it from the production line and place it in a reject bin. This process would require a very keen eye from several humans in order to visually inspect every container accurately, but the MV system does this with ease.

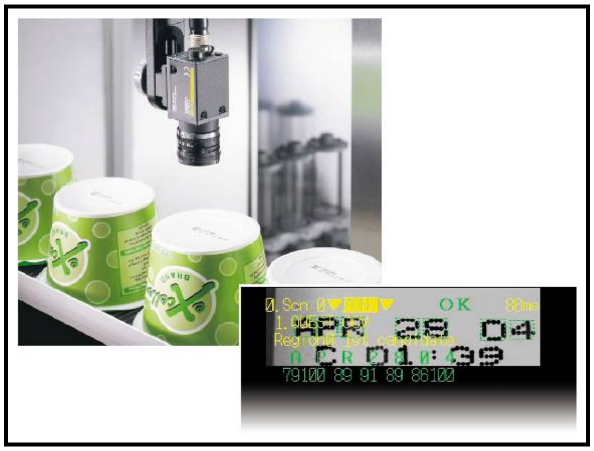


Figure 2.2: MV Inspection System for Date and Time on Yogurt Cups[49]

K-Means Clustering Algorithm is attractive in that there is only one user definable input and that is the number of clusters to be found. As a result of this, the K-Means algorithm is very popular both in research and in industry. This can be seen in a paper presented by Ramos and Muge of Portugal where the standard K-Means Algorithm was used to segment maps[52]. They used K-Means because, according to the researchers, segmenting a color image composed of different kinds of texture regions can be a hard problem[52]. By using the K-Means Clustering Algorithm, their segmentation problem was workable by simply knowing how many different textures there were on the map. Future research is now being performed using an adaptive method of determining the number of textures in the image instead of needing a human input.

In another application, the Hough Transform has been used to detect the borders in patterned fabric[12]. Combined with the use of the FUZZY C-Means Algorithm mentioned earlier, the Hough transform accurately detects the lines that make up the borders of the regions in the printed fabric. The importance of this application is far removed from the importance seen with the use of the Hough Transform in medical imaging, but it is a point that should be stressed and that is the fact that in almost any application involving the human eye, MV can step in and do a very remarkable job of replacing the human.

2.3.4 Medical Industry/Biology

As previously stated, the medical industry is on the forefront when it comes to using imaging in a critical process. The medical industry uses imaging in almost all diagnostic procedures either in the form of a computer tomography (CT) scan, ultrasound, or magnetic resonance imaging (MRI). These advances in technology have come about in the last two decades and have made diagnostic medicine much more reliable with the ability to see what is happening inside the human body. MV is also not exclusively applied to diagnosis purposes. MV is also used to help blind people read or semi-blind people to see major objects. There is much research being performed in this area where helping the handicapped is the main objective.

An example of research being performed to help the handicapped is seen in research that has been conducted by Ferreira, Garin, and Gosselin at the Faculte Polytechnique de Mons in Belgium[53]. This research focused on text detection in many situations, but in all cases in order to single out the text, thresholding was used to simply filter out the background and emphasize the text so that the image could be converted to a binary form. Then, a more advanced region based technique was used to pick out each letter and essentially 'read' the text. This is another example of thresholding being used as a pre-process to a more advanced technique.

Other research has focused on a different set of objectives. According to researchers in the United Kingdom, many people with vision problems resulting in "low vision" such as having cataracts, diabetic retinopathy, age-related maculopathy, and retinal detachment are not totally blind, but they retain some residual vision[54]. This residual vision is usually not enough to allow mobility of the person, but the researchers have used the K-Means Clustering Algorithm to pick out major objects in a FOV and then display them in a head mounted display which would show much less detail than a normal scene would as viewed by a person with low-vision[54]. This would allow them to pick out objects more easily without the "noise" created by all of the details. The end result is that the person, who was not previously mobile, could now move around with the ability to see main objects in their field of vision without being confused or blurred by the details of the entire image.

As discussed before, MV is extremely important to the functioning medical imaging and it has been found that the K-Means Clustering Algorithm is also a very important subtopic in the use of MV in medical imaging. In a presentation highlighting the segmentation methods available in segmenting a moving organ in a CT Scan, it was shown that among all of the available segmentation techniques/algorithms, the K-Means Clustering Algorithm was more effective in speed and in detecting subtle differences among pixels that highlighted different regions in the organ[55].

Mark Dow of University of Oregon has also completed research[56] in the area of neurosciences dealing with edge-based segmentation. As mentioned earlier, the medical field is the forefront in image segmentation research. The research performed by Dow deals with detecting the borders between white matter and gray matter in the brain from images taken with a MRI. This research is more flexible in what can be segmented in the images, but it is important that images with low spatial frequency be used so that the determination between segments is not a hard decision for the algorithm to make. It can be seen that almost unconditionally, these techniques are coupled with some other algorithms to achieve the final goal. It is stressed that this type of low-level edge-based segmentation is just that: low-level, and as such, generally requires additional algorithms to achieve the final goal.

As with almost every aspect of MV, the Hough Transform has been applied to the medical imaging area as well. In a paper written by researchers at Texas Tech University, the Hough Transform was used to detect cervical vertebrae in x-ray images[57]. Not only did this approach work but also it was very robust in detecting bone fragments and anomalies on individual vertebrae that would have otherwise been hard for a physician to detect by eye. The robustness of this algorithm in this application is a very desirable trait since bone growth is something that can vary greatly from individual to individual. Therefore, the likelihood that an error in diagnosis being made is even further reduced using the Hough Transform.

One last example of thresholding being used occurs in the biology discipline. Researchers at the University of California at San Francisco recently used MV techniques to study behavioral patters of mutant worms[58]. By first imaging the worm in monochrome, thresholding was applied in order to convert the image to binary so that the worm could be easily distinguished from the background. Again, higher-level segmentation techniques were then applied to be able to measure and track the worm's movements to determine its behavioral patterns.

2.3.5 Agriculture

Another area of research that has been applied to industry, currently using the K-Means Clustering Algorithm, is in the agriculture industry. Researchers at the University of Illinois have applied the K-Means Clustering Algorithm in order to detect weeds in real-time, as the herbicide spraying machine was making its way down a row of soybeans[59]. The machine is able to count, classify, and then spray each weed individually so that a minimal amount of herbicide is used and so that the effectiveness of the herbicide can be evaluated and tracked by noting the location of the weed. Then, in a subsequent spraying operation, each particular weed can be evaluated as to whether it was killed or not, further enhancing the ability of the farmer to pinpoint specific types of weeds in his field and eradicating them. The results of their research has shown this application of MV be both a very cost effective and time conserving way to do this important job in extremely large plots of farmland and by doing so, farmers have increased crop yield and decreased ground water contamination.

Chapter 3

Theoretical Approach

3.1 Overview of Theoretical Approach

The theoretical approach to the problems presented here can be broken down into two distinct layers. The first layer includes low to medium level image processing functions used in the pre-processing stage, on an as needed basis, in order to work toward a solution to the problems of Marker Detection and Tracking and Runway Detection. These functions do not differ in their inclusiveness to each problem, but the sequence in which they were used in order to achieve an acceptable result may be different and these differences will be fully explained in Chapter 4. Therefore, these functions, known as *Shared Image Processing Functions*, will be discussed in Section 3.2. Section 3.2 highlights the theory behind these functions and also covers their application to both problems in order to eliminate repeating the theoretical discussion regarding these functions for each problem individually. The second layer involves high level image processing methods and algorithms which are used to perform the main task needed to solve each problem such as line-detection or marker detection and tracking. These methods and/or algorithms are unrelated as they apply to each individual problem, hence, they will be discussed separately in Section 3.3 and Section 3.4, respectively.

3.2 Shared Image Processing Functions

As previously mentioned, this section will cover functions that were used in the pre-processing stage of both the Marker Detection and Tracking and Runway Detection solutions. The fact that these functions were used to approach both problems reflects the versatility of the low to medium level image processing techniques and emphasizes a statement made earlier that many image processing solutions and, in this case, subsystems such as the pre-processing stage, are just a collection of smaller, lower level processes. These processes include the most basic things such as the definition of the coordinate system and an image, which are covered in Section 3.2.1 and Section 3.2.2. Section 3.2.3 covers image enhancement functions such as Gamma Correction and Color

Space Conversions. Lastly, Section 3.2.4 presents the concept of and theory behind thresholding in image processing.

3.2.1 Coordinate System

First, a spatial coordinate system must be defined. The nomenclature *f(x,y)* will be used to define a point in a two-dimensional image frame, where *x* and *y* denote spatial coordinates and the value of *f* at any point *(x,y)* is proportional to a color level value normally ranging from 0 to 255 when speaking of an image constructed of separate red, green, and blue (RGB) values. Figure 3.1 illustrates the coordinate convention used during image processing.

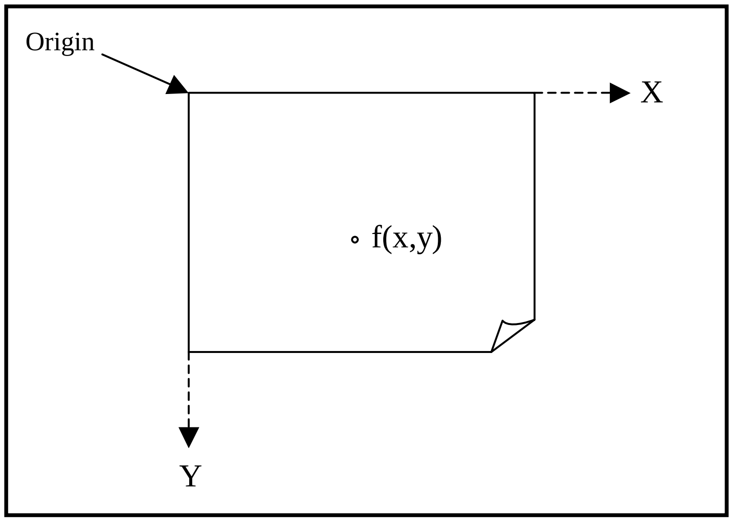


Figure 3.1: Coordinate Convention for Images

Suppose that a continuous image is sampled uniformly into an array of *N* rows and *M* columns, where each sample represents a color level value. This array is subsequently called a digital image and is represented by Equation 3.1:

$$f(x,y) = \begin{bmatrix} f(1,1) & f(2,1) & \ldots\ldots. & f(M,1) \\ f(1,2) & f(2,2) & \ldots\ldots. & f(M,2) \\ \ldots\ldots. & \ldots\ldots. & \ldots\ldots. & \ldots\ldots. \\ f(1,N) & f(2,N) & \ldots\ldots. & f(M,N) \end{bmatrix} \qquad (3.1)$$

where *x* and *y* are discrete values: *x = 1,2,3,…,M; y = 1,2,3,…,N*. Each element in the array is defined as a pixel.

3.2.2 Image Definition

An RGB image is composed using a red, green, and blue part which are stacked on top of one another. The resulting color is a combination of the three colors at each spatial location, resulting in a blended color with more than 16.7 million variations using the standard 0 to 255 color pallette (8-bit). Sometimes, decimal values from zero to one will be used to represent the 8-bit color pallette. This type of RGB image definition is simply another way to define the image and is commonly used in Matlab®. This type of image definition is illustrated in Figure 3.2.

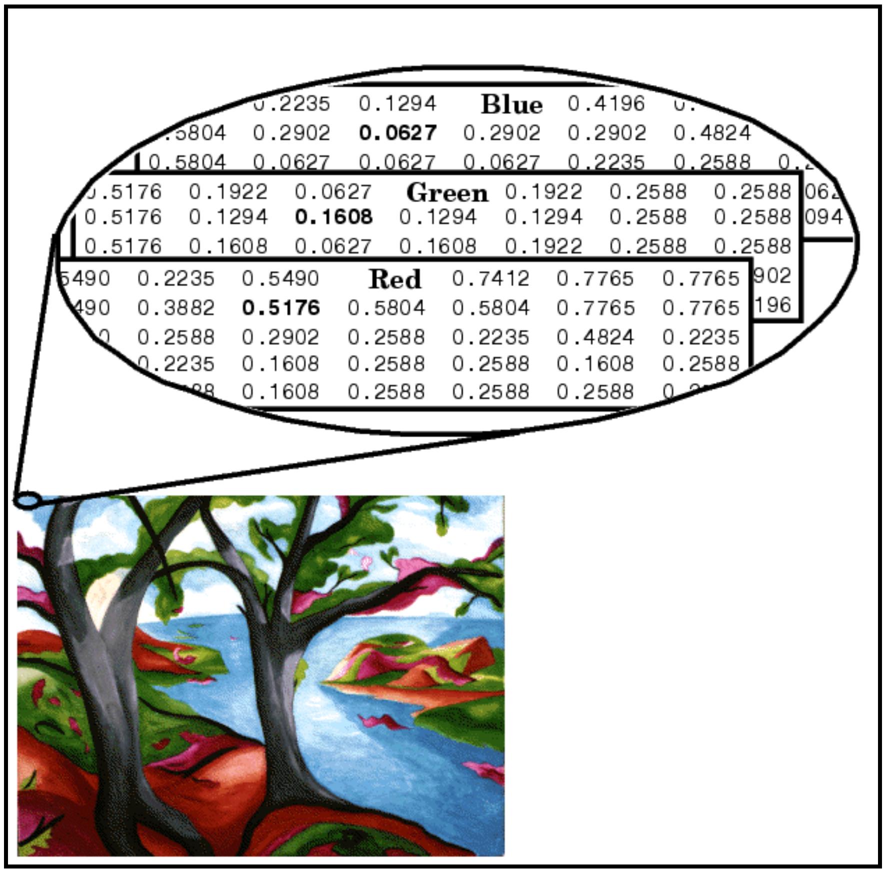


Figure 3.2: RGB Image Construction[60]

In most cases, the image is based on a color map, which may have any range of values. This range of values will correspond to a certain combination of RGB values that create the actual pixel color. Although this arrangement is not directly seen unless a distinct color map is defined, this is the process that is happening behind the scenes but it is most likely happening using a 'standard' 8-bit color map. Although, a custom color map is not usually defined, sometimes this can be useful if the user requires many variations in one shade of a color. It is possible to define an infinite number of shades of any color and then use them to construct an image using the color map image definition method. This concept is illustrated in Figure 3.3.

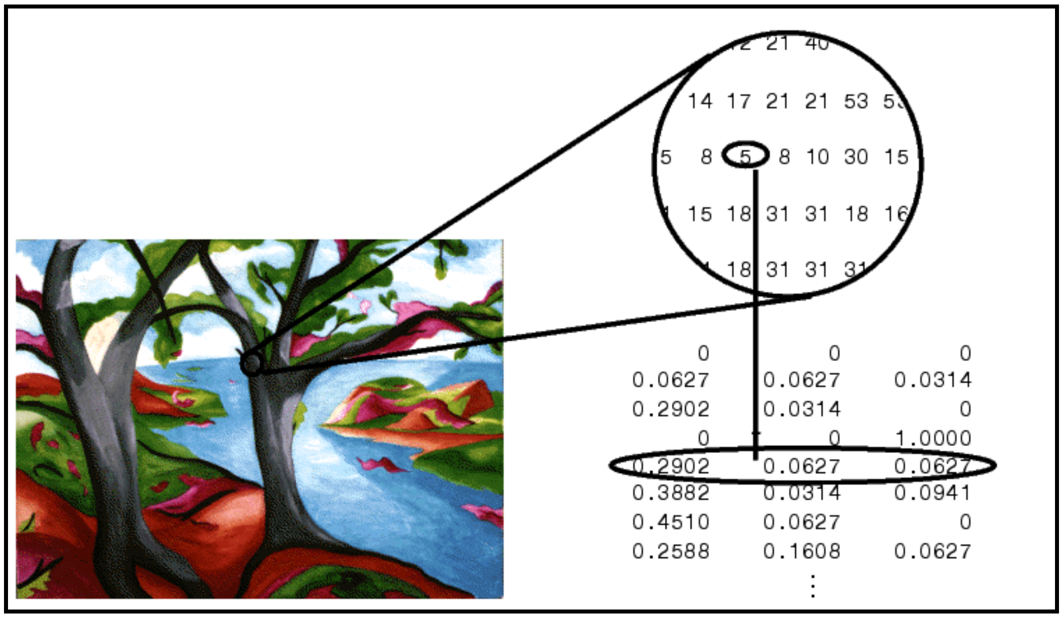


Figure 3.3: Color Map Style of RGB Image Definition[60]

The value of *f* at any point *(x,y)* can also be proportional to a brightness level value ranging from zero to one when speaking of values in an intensity image. There are several distinct differences between an RGB image and an intensity image. First, the intensity image never uses any sort of color map. Second, the intensity image is made up of only one image or matrix, instead of three. Speaking on terms of similarities, there is no limit on the discretization of the values except restrictions put on the image by the software platform. An intensity image is similar to having an image made up of only red, green, or blue except the 'color' is equivalent to brightness, ranging from white to black. This essentially creates a grayscale image except that the colors of the image or not shades of gray but shades of pure brightness ranging from black to white. Figure 3.4 illustrates the concept of an intensity image with values of class *double*.

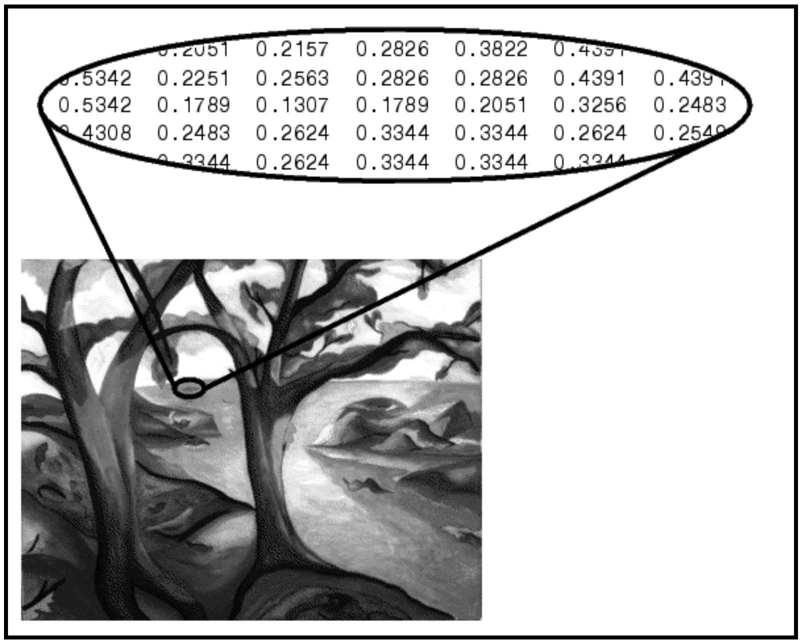


Figure 3.4: Illustration of the Construction of an Intensity Image[60]

3.2.3 Gamma Correction and Color Level Conversion

Gamma correction is a relationship between an image having linearly increasing intensity and an image having linearly increasing luminance. Gamma correction is usually performed in consumer video systems such as televisions and video cameras. But, in MV, this aspect is usually left up to the system designer. In this research, the use of gamma correction was a necessity rather than a want. All Matlab® functions involving RGB images required the image to be gamma corrected. Therefore, gamma correction was the first step during the processing sequence and will be explained in Section 3.2.3.1.

Color level conversion was also an important first step for this image processing application. When using this conversion within Matlab®, it is usually a second step due to the fact that Matlab® requires the input to the conversions to be gamma corrected. Therefore, the color level conversion usually takes a back seat to the gamma correction for this reason. Color level conversion offers the ability to reduce computational workload by representing the image in a different form. It has the ability to maintain high amounts of detail in an image while representing it in a different format. The color level conversion method used in this research effort will be explained in Section 3.2.3.2.

3.2.3.1 Gamma Correction

The gamma characteristic is a power-law relationship that approximates the relationship between the encoded luminance in a video system and the actual desired image brightness. With this non-linear relationship, steps in encoded luminance correspond to subjectively approximate steps in brightness[61]. MV systems and software that require a linear relationship between these quantities, such as the Matlab® environment, use gamma correction. Although the gamma correction could have been performed in software, there was a second option available in this research. The MV camera used in this research had the ability to perform hardware gamma correction internally. This method of gamma correction was used in this research effort in order to further reduce the steps in the pre-processing stage. Equation 3.2[61] represents the general form for hardware based gamma correction:

$$I = V_S^{\gamma} \tag{3.2}$$

where I is the light intensity, V_S is the source voltage coming from each pixel location in the charge coupled device (CCD), and γ is the gamma correction factor.

Gamma correction can be thought of as an inverse transfer function that is applied to the video signal so that the encoded luminance is linear. The following illustration, Figure 3.5, shows the difference between a scale with linearly increasing intensity (i.e. gamma corrected) scale and a scale with the desired linearly increasing encoded luminance signal[61].

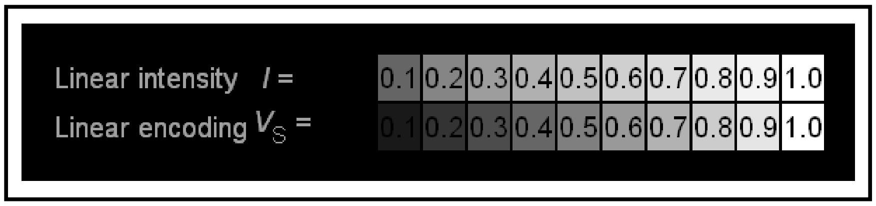
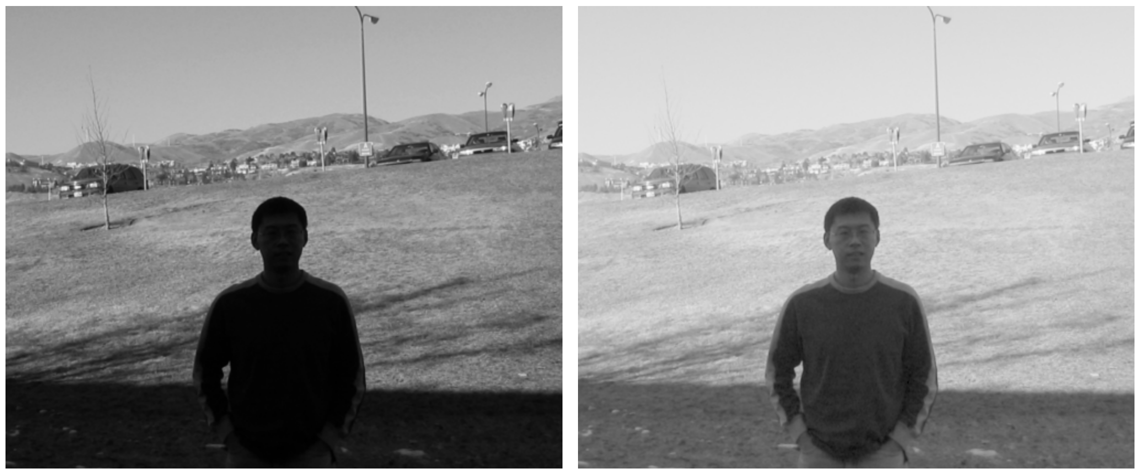


Figure 3.5: Gamma Correction Example Scale[61]

In layman's terms, the signal from the video camera or source is deliberately distorted so that after it has been distorted again by the display device (in this case the framegrabber card and video card), the viewer (Matlab® in this case) sees the correct brightness. It should be noted that from this point on, a normal, non-gamma corrected signal will be referred to as the representative letter, such as RGB, and a gamma corrected signal will be referred to with an added prime symbol, such as $R'G'B'$.

Figure 3.6, below, shows a visual example of what impact gamma correction can have on images. Figure 3.6a shows an image that is taken in bad lighting where the gentleman's face cannot easily be seen. Figure 3.6b shows a gamma corrected version using a gamma correction value of 2.25 and now, the gentleman's face is clearly defined. This can be useful in aerospace applications where the lighting may not be suitable to extract the details needed from the image to continue the processing task. A simple gamma correction can fix this and, in this research effort, the hardware based gamma correction was used to approach the problems. This gamma correction served a dual purpose in helping to satisfy the Color Space Conversion inputs by providing them with the required gamma corrected image and it helped to brighten up the image when it was dark or overcast.



(a) Original Image **(b)** Gamma Corrected Image

Figure 3.6: Comparison of Original Image to Gamma Corrected Image[62]

3.2.3.2 RGB to Intensity

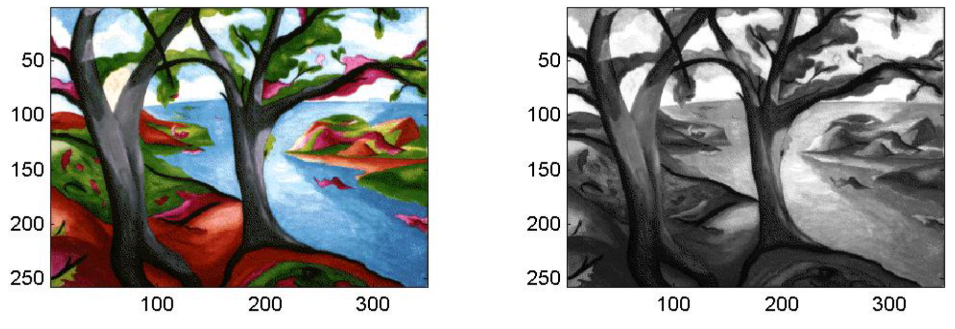
Converting an RGB image to an intensity image has obvious advantages in changing an image described by a M×N×3 matrix to an image described by an M×N×1 matrix. Through a reduction in the size of the third dimension, the complexity of the image definition is greatly reduced.

This conversion is described by a mathematical equation involving the intensity of each red, green, and blue pixel of a point of interest. To find the intensity of a pixel in gray level, the following formula, Equation 3.3[60], is used.

$$\text{intensity} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} \tag{3.3}$$

where R', G', and B' are the gamma corrected color level values for each respective pixel of the original RGB image.

The range of any input pixel value will match that of the output pixel intensity value. The illustration shown in Figure 3.7 shows an example of the input and output of the RGB to Intensity function of the function. The input image is shown in Figure 3.7a and it is a typical M×N×3 RGB image. The output image is shown in Figure 3.7b and it has been converted to a M×N×1 intensity image.



(a) Original Image **(b)** Intensity Image

Figure 3.7: Example Images for the R′G′B′ to Intensity Color Conversion[60]

3.2.4 Thresholding

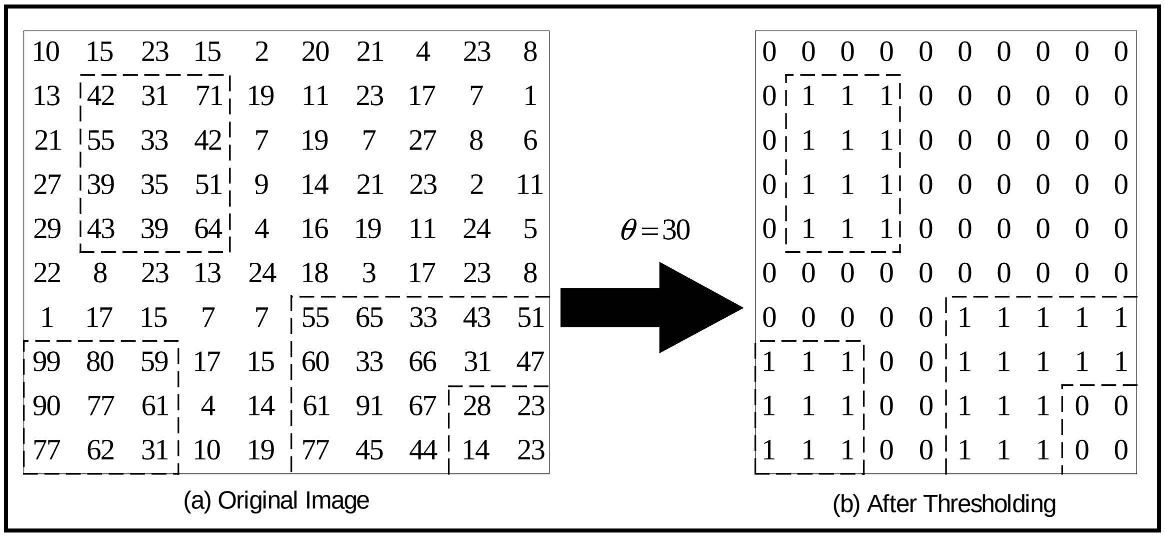
Thresholding is a simple process that is also a very valuable filtering technique in image processing. Thresholding has many meanings and many purposes, but probably the most widely used application is in filtering out a certain color or shade from an image. Many images may contain things that are unwanted and many times these things are homogeneous in the image, such as a grassy field. Thresholding has the ability to find all of the pixels that are green and set them to be another color such as black, that will be ignored by other algorithms. This is a very simple but efficient form of image segmentation.

Thresholding can also have another meaning when talking about gray scale images. Sometimes the threshold level is referred to as a percentage. This percentage of thresholding means the threshold level between the maximum and minimum intensity of the initial image. Thresholding is a way to get rid of the effect of noise and to improve the signal-noise ratio if the noise is homogeneous in intensity. To put this in laymen's terms, it is a method that allows the user to keep the significant information of the image while disposing of the unimportant part (under the condition that is chosen as a plausible thresholding level). The use of thresholding will be fully evident later when the use of thresholding is shown in the research software being used for the purpose of image segmentation.

Perhaps the easiest explanation of thresholding is mathematically. Once the mathematical definition is made, it is easy to find many different areas to apply the concept of thresholding to. Equation 3.4, below describes the thresholding process:

$$\begin{array}{ll} \text{If } a[m,n] \geq \theta & a[m,n] = object = 1 \\ \text{Else} & a[m,n] = background = 0 \end{array} \qquad (3.4)$$

where *a* is the image defined by the pixel coordinates *m* and *n* and theta is the threshold value. This method assumes that the interest lies in light objects on a dark background. If a pixel value is greater than a certain threshold value, the pixel value is changed to one or white, if it is less than the threshold, the pixel value is made to be zero or black. This is simple image segmentation. Figure 3.8, below, illustrates a simple thresholding performed on an image for segmentation purposes. Figure 3.8a is the input image and Figure 3.8b is the output image.



10	15	23	15	2	20	21	4	23	8
13	42	31	71	19	11	23	17	7	1
21	55	33	42	7	19	7	27	8	6
27	39	35	51	9	14	21	23	2	11
29	43	39	64	4	16	19	11	24	5
22	8	23	13	24	18	3	17	23	8
1	17	15	7	7	55	65	33	43	51
99	80	59	17	15	60	33	66	31	47
90	77	61	4	14	61	91	67	28	23
77	62	31	10	19	77	45	44	14	23

0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1
1	1	1	0	0	1	1	1	1	1
1	1	1	0	0	1	1	1	0	0
1	1	1	0	0	1	1	1	0	0

Figure 3.8: Illustration of Thresholding for Image Segmentation Purposes

3.3 Theoretical Approach to the Marker Detection and Tracking Problem

Aside from decisions regarding the pre-processing stage of the solution, one must decide how to actually find the markers for the Marker Detection and Tracking problem. Image segmentation can be described as the process of separating the useful information of an image from the non-useful information. The problem of Marker Detection and Tracking falls into the category of an image segmentation problem. After reviewing several methods available, looking at how others have approached similar problems, consulting with people working in image processing, and taking into account design constraints discussed earlier in Section 1.2.1 it was decided that the problem of Marker Detection and Tracking would be approached with the K-Means Clustering Algorithm. The first solution was approached using Matlab® and it involves the use of a Modified K-Means Clustering Algorithm. This algorithm is called the Modified K-Means Clustering Algorithm because it was modified to make it more efficient in scanning a large number of pixels at a high frame rate. This will be covered in more detail in Section 3.3.1. The second solution was also approached using Matlab®. This solution involves the use of the Modified K-Means Clustering Algorithm and an additional set of instructions that estimates and tracks the positions of each marker. This solution will be covered in Section 3.3.2.

3.3.1 Matlab Based Modified K-Means Clustering Algorithm

For a description of the mathematical representation of the K-Means Clustering Algorithm, the following equations are presented. First, a measure of similarity must be established which will determine if pixels are assigned to the domain of a particular cluster. For this algorithm, the Euclidean distance, d, between two pixels, m and n is used as a measure of similarity and this is shown below in Equation 3.5 through Equation 3.7[50]:

$$d = \| m - n \| \tag{3.5}$$

$$d = \sqrt{(m_x - n_x)^2 + (m_y - n_y)^2} \tag{3.6}$$

$$d = \sqrt{\sum_{k=1}^{n} (m_k - n_k)^2} \tag{3.7}$$

where m and n are n-dimensional vectors with k components equal to m_k and n_k, respectively.

In order for the algorithm to determine which cluster a new pixel belongs to, a performance index must be introduced. The clustering criterion is based on the minimization of the performance index that is guided by a procedure that will minimize or maximize the result of the similarity measure, d. The performance index, J, is the sum of the errors index given below in Equation 3.8[50]:

$$J = \sum_{j=1}^{N_c} \sum_{x \in s_j} | x - m_j | \tag{3.8}$$

where N_c is the number of clusters, S_j is the set of samples belonging to the j_{th} domain, x is the data point to be clustered, and

$$m_j = \frac{1}{N_j} \sum_{x \in S_j} x \tag{3.9}$$

is the sample mean vector of the set, or the center of cluster S_j. In Equation 3.9, N_j represents the number of samples in cluster S_j. The index of Equation 3.8 represents the overall sum of the errors between the samples of a cluster domain and their corresponding mean.

The K-Means Clustering Algorithm consists of the following steps.

1. Scan the image frame until a point is accepted (the first white pixel). Set this point as the initial cluster center, Z_1, and cluster center S_1.

2. Scan the image again for the next white pixel. Set it as a new point, X .

3. Computer absolute distance, d_i, from new point X to previous cluster center, Z_i, and distribute the point { X } among the cluster domains using minimum distance similarity,

$$X \in S_i \text{ if } | X - Z_i | < T \tag{3.10}$$

for all $i = 1, 2, 3, ..., k$, where S_i denotes the set of points whose cluster center is Z_i and T is a predetermined minimum distance threshold for similarity.

4. If the new point satisfies the condition in *Step 3*, then go to *Step 5*, else cluster the new point as a new cluster group and a new center,

$$S_{i+1} = X \quad \text{and} \quad Z_{i+1} = X \tag{3.11}$$

5. Count the number of points in each cluster group and store it in an array N_i for $i = 1, 2, 3, ..., k$.

6. Sum the point locations for each cluster group and store it in array SUM_i for $i = 1, 2, 3, ..., k$,

$$SUM_i = \sum_{i=1}^{k} X_i \tag{3.12}$$

7. From the results of *Step 5* and *Step 6*, compute the new cluster centers Z_i , such that the absolute distances from all points to the new cluster center is minimized.

$$Z_i = \frac{SUM_i}{N_i} \tag{3.13}$$

8. If it is the last row of the image frame, go to *Step 9*, otherwise go to *Step 2*.

9. Stop.

The K-Means Clustering Algorithm was determined to be a feasible algorithm to formulate an approach to the problem of Marker Detection. Once the decision to continue with the K-Means Clustering Algorithm was made, it was examined in more detail with respect to its application in a vision problem. When considering the aspect of efficiency it was noted that the K-Means Clustering Algorithm is a fairly slow process by design because it requires many, many complete scans of the image to find the pixels of interest. While finding the pixels of interest, the algorithm is constantly grouping the pixels based on a distance threshold, which further slows the process down. It then calculates the current centroid for the group in question. It then uses these calculated centroids to compare against the distance threshold for future decisions. It continues this process of scanning, grouping, and recalculating the centroids for each individual white pixel that is encountered in the image. This algorithm works fine when the ability of running in near real-time is not desired but this is not the case with the software in this research effort. Therefore, modifications were made to the way the K-Means Clustering Algorithm is performed. This will be called the Modified K-Means Clustering Algorithm.

The Modified K-Means Clustering Algorithm is an algorithm based on the K-Means Clustering Algorithm but changes have been made which allows the algorithm to be much more efficient. The main reasoning behind these changes can be explained by examining the number of iterations required to scan the image one time. An image that is of 640×480 pixels in resolution contains just over 300,000 pixels. The original K-Means Clustering Algorithm performed multiple scans of these 300,000 pixels to achieve the clustering task. This multiple scanning of the image resulted in extra work and computational effort required by the computer and no reasoning behind this approach could be found. Therefore, the algorithm was modified in such a way that the image was scanned only once per frame of input data. This modification greatly improved the speed of the algorithm because most of the time in image processing is spent scanning the image for useful information. In order to streamline the K-Means Clustering Algorithm further, the constant calculation of the centroid of the group when a new pixel is added to the group was abandoned. In order to calculate the centroid, two squaring functions and a square root function were needed. Since the square root is performed by iterative numerical methods such as Newton’s method, it is widely known as one of the most burdening computations for a computer to perform. This constant calculation of the square root would continuously use this computational burdening square root function and this was found to be impractical and the constant calculation of the centroids was abandoned. Once this centroid calculation was abandoned, the algorithm began to take on a different shape as new, more efficient ways to do these jobs were developed.

A side-by-side, step-by-step comparison chart of the original K-Means Clustering Algorithm versus the Modified K-Means algorithm is presented in Table 3.1, on the following page.

Table 3.1: K-Means Algorithm vs. Modified K-Means Algorithm

	K-Means Clustering Algorithm	**Modified K-Means Algorithm**
Step 1	Scan the image until a white pixel is found, assign point as cluster center Z_1 and cluster group S_1.	Scan entire image and compile a list of all of the white pixels in the image. Initialize the first cluster group by assigning the first pixel in the list to the first group.
Step 2	Scan the image again, find the next white pixel, and set it as X. Compute absolute distance from new point X to all previous cluster centers Z_i and apply a minimum distance threshold.	Examine the next pixel in the list and compare its X and Y coordinate to the X and Y coordinate of the last pixel encountered based on a threshold.
Step 3	If the new point satisfies the minimum distance threshold for a cluster Z, it is added to that cluster list and a new centroid is computed.	If the new point satisfies the threshold condition, it is added to the pixel list for the cluster in question.
Step 4	If not, the point is the added to a new cluster group and a new cluster center is defined.	If not, it is defined as a new cluster group and the process continues until all of the pixels in the list of white pixels have been evaluated.
Step 5	Return to Step 2.	Once the lists of points belonging to each cluster have been compiled, the centroid of each cluster group is calculated.
Step 6	If no more pixels are found, Stop.	Stop.

It can be seen from this comparison that there are some major differences. The most obvious difference is that the image is now only being scanned once per image frame instead of multiple times. The second difference is that the centroids are no longer constantly calculated as the algorithm progresses. It was determined that this was not necessary because the image scanning is performed left to right and the grouping steps use the first pixel encountered, which is the leftmost pixel in a group. So, instead of going through the trouble of computing the centroid, the threshold is simply applied to the right hand side of the left most pixel in the group and it is ensured that the threshold is big enough to encompass all of the pixels in the group of interest. This concept relies on the assumption that the camera is always in focus, but greatly reduces the computational complexity of the algorithm. When the camera is out of focus, the marker will become blurred and possibly appear large enough to exceed the threshold boundary.

3.3.2 Matlab Based Advanced K-Means Clustering and Tracking Algorithm

Once the Modified K-Means Clustering Algorithm was created and evaluated, an idea involving the tracking of each marker and the estimation of the position in the next frame presented itself and it was thought that this would make the algorithm faster and more efficient. This idea became the *Advanced K-Means Clustering and Tracking Algorithm* and it involves finding the markers in the first few frames of the video with the Modified K-Means Clustering Algorithm, then by looking at the marker positions in the past few frames, an estimation of the position of the markers in the next frame is performed. From engineering dynamics, only three points are needed to estimate the position of a point based on the velocities and acceleration. By doing this, the computation speed is greatly increased and these results will be compared and discussed in Chapter 5.

The solution begins by finding the markers from three consecutive frames using the same method as in the Modified K-Means Clustering Algorithm. Once these marker positions are found, the velocity of each marker is determined by Equation 3.14, below:

$$V = \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \dfrac{x(index) - x(index-1)}{\Delta t} \\ \dfrac{y(index) - y(index-1)}{\Delta t} \end{bmatrix} \tag{3.14}$$

where x is the X-coordinate of the centroids of the markers and y is the Y-coordinate of the centroids of the markers at frame number $index$. Δt is the time in seconds between the two sequential frames. V_x and V_y are the velocities in the x and y directions, respectively. Figure 3.9, on the following page, represents the marker position situation for calculating velocity.

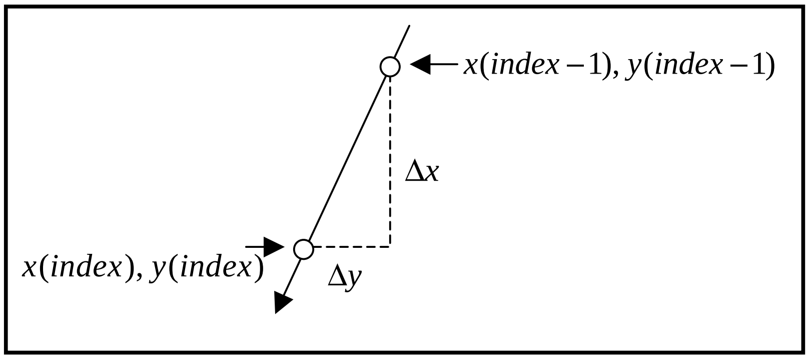


Figure 3.9: Marker Position Situation for Calculating Velocity

Once the velocities were found, Equation 3.15, below, was used to calculate the estimated position of the markers in the next frame:

$$\begin{bmatrix} x(index+1) \\ y(index+1) \end{bmatrix} = \begin{bmatrix} x(index) \\ y(index) \end{bmatrix} + \begin{bmatrix} V_x(index) \\ V_y(index) \end{bmatrix} \Delta t \tag{3.15}$$

The final step in this solution is, of course, incorporating the acceleration into the process. The acceleration of the markers can be found using Equation 3.16 and the expanded version, Equation 3.17:

$$A = \begin{bmatrix} A_x \\ A_y \end{bmatrix} = \begin{bmatrix} \dfrac{V_x(index) - V_x(index-1)}{\Delta t} \\ \dfrac{V_y(index) - V_y(index-1)}{\Delta t} \end{bmatrix} \tag{3.16}$$

$$A = \begin{bmatrix} A_x \\ A_y \end{bmatrix} = \begin{bmatrix} \dfrac{x(index) - x(index-1)}{\Delta t_1^2} - \dfrac{x(index-1) - x(index-2)}{\Delta t_2^2} \\ \dfrac{y(index) - y(index-1)}{\Delta t_1^2} - \dfrac{y(index-1) - y(index-2)}{\Delta t_2^2} \end{bmatrix} \tag{3.17}$$

Now, the estimated marker location in the next frame becomes Equation 3.18:

$$\begin{bmatrix} x(index+1) \\ y(index+1) \end{bmatrix} = \begin{bmatrix} x(index) \\ y(index) \end{bmatrix} + \begin{bmatrix} V_x(index) \\ V_y(index) \end{bmatrix} \Delta t + \tfrac{1}{2} \begin{bmatrix} A_x(index) \\ A_y(index) \end{bmatrix} \Delta t^2 \qquad (3.18)$$

Figure 3.10 illustrates the estimate of a new marker location using velocity and acceleration.

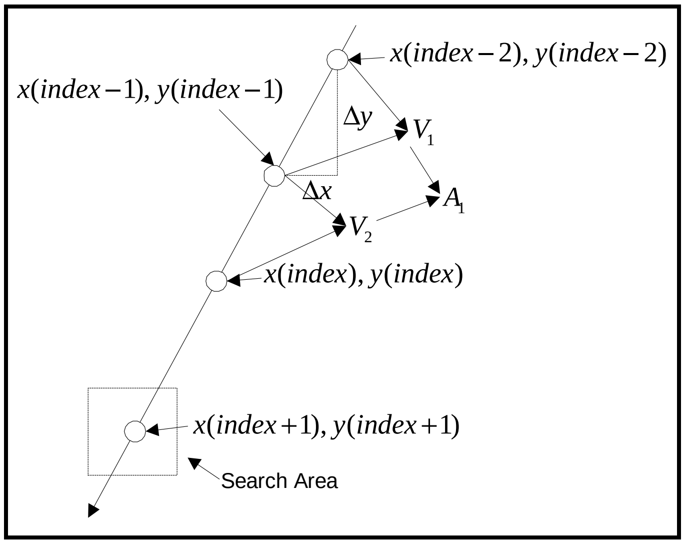


Figure 3.10: Estimate of New Marker Location Using Velocity and Acceleration

3.4 Theoretical Approach to the Runway Detection Problem

The problem of Runway Detection requires an entirely different way of thinking than does the problem of Marker Detection and Tracking. The methods used to approach the Marker Detection problem are general algorithms adapted to perform the job of Marker Detection. Since the Runway Detection problem is a more complex problem, the same architecture and use of non-specialized algorithms will not be sufficient to approach this problem because of the many variations of the image that could be presented to the algorithm because of the more uncontrolled environment that is encountered in this application. Runway Detection is a problem which similarly involves filtering, but aside from that, the presence of a much higher-level problem exists. This problem consists of deriving information relating to the location of a runway in an image frame from an entirely homogeneous color image. This intensely complicates matters, especially for the filtering. Given the complexity of this problem, Simulink® and the Video and Image

Processing Blockset® [60] and the Image Acquisition Toolbox® [63] was chosen solely for the purpose of solving this problem.

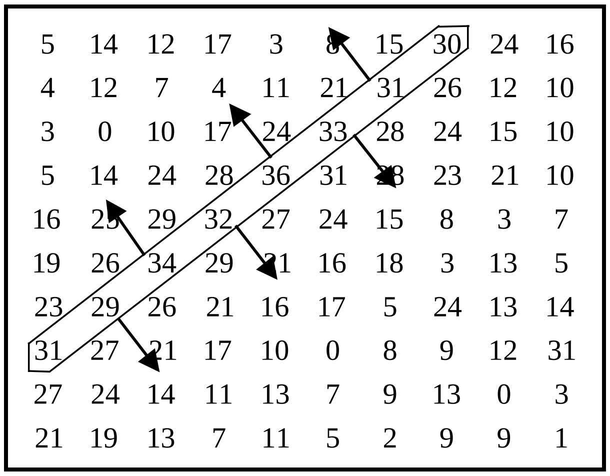
There are essentially three processes that are required for this solution which are above and beyond the pre-processing functions previously mentioned in Section 3.1. The first is a medium-level image processing function called 'edge detection'. There are several different types of edge detection routines but only one of them was needed for this research effort and it is covered in Section 3.4.1. The second is a medium-level image processing function called 'morphological opening'. Morphological opening consists of two children functions called morphological dilation and morphological erosion. All three of these functions will be covered in Section 3.4.2. The third is a high-level image processing function known as the Hough transform. This is the basis for the solution to the runway detection problem and it is covered in Section 3.4.3.

3.4.1 Sobel Edge Detection

Edge detection is one of the most important fundamental operations in image processing and many applications rely solely on edge detection and, thus, would not be possible without it. There are two main types of edge detection algorithms: Gradient Based and Laplacian Based[64]. The gradient based algorithms can be further broken down into three algorithms: Sobel, Roberts, and Prewitt[64]. In looking at a line in the gradient frame of mind, the values leading up to an edge and following an edge will always increase and then decrease. This is true in a grayscale image or a binary image. As a result of this, these gradient-based methods all use the same approach but they use different convolution matrices.

The Laplacian based algorithms only consist of one algorithm that fits this description: that is the Canny edge detector[64]. The Canny edge detector finds edges by looking for the local maxima of the gradient of the input image, which it calculates from the derivative of the Gaussian filter[64]. The three gradient based methods are very similar to each other, so similar in fact, that most of the time the eye cannot detect the difference in the lines that have been detected and almost 100% of the time, the computer software will not perform any differently using any of the three methods. For this reason, only the most common Sobel operator will be discussed in this section.

The Sobel edge detector is a gradient-based edge detection operator. The Sobel operator performs a 2-D spatial gradient measurement on an image and as such, it emphasizes regions of high spatial frequency that correspond to edges[65]. Regions of high spatial frequency correspond to edges because an edge is not usually made up of a single line of pixels. Rather, it is made up of a group of pixels whose intensity increases and decreases as the ‘line’ approaches, much like a car does when it is traveling over a mountain. The peak represents the real line and the slope leading up to it is usually what needs to be filtered out. This mountain peak analogy can be seen in Figure 3.11, below.



5	14	12	17	3	8	15	30	24	16
4	12	7	4	11	21	31	26	12	10
3	0	10	17	24	33	28	24	15	10
5	14	24	28	36	31	28	23	21	10
16	25	29	32	27	24	15	8	3	7
19	26	34	29	21	16	18	3	13	5
23	29	26	21	16	17	5	24	13	14
31	27	21	17	10	0	8	9	12	31
27	24	14	11	13	7	9	13	0	3
21	19	13	7	11	5	2	9	9	1

Figure 3.11: Illustration of Line Definition in a Typical Image

The Sobel operator consists of a set of convolution kernels. These kernels are designed to bring out the vertical and horizontal gradients separately. The definition of these kernels are shown in Equations 3.19[65] and 3.20[65], on the following page:

$$G_X = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{3.19}$$

$$G_Y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.20}$$

As stated before, these kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation. This is the exact method that Matlab® uses. The resultant images from applying the vertical kernel and the horizontal kernel to a test image of some grains of rice is seen on the following page in Figure 3.12a and Figure 3.12b, respectively. These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by the following equation, Equation 3.21[65]:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.21}$$

where G_x and G_y are the individual resultant image matrices associated with each kernel and $|G|$ is the total gradient magnitude matrix. It can be seen in Figure 3.12a that the vertical lines are darker and more well defined than the horizontal lines. The same is true with Figure 3.12b with the horizontal lines being darker and more well defined than the vertical lines.

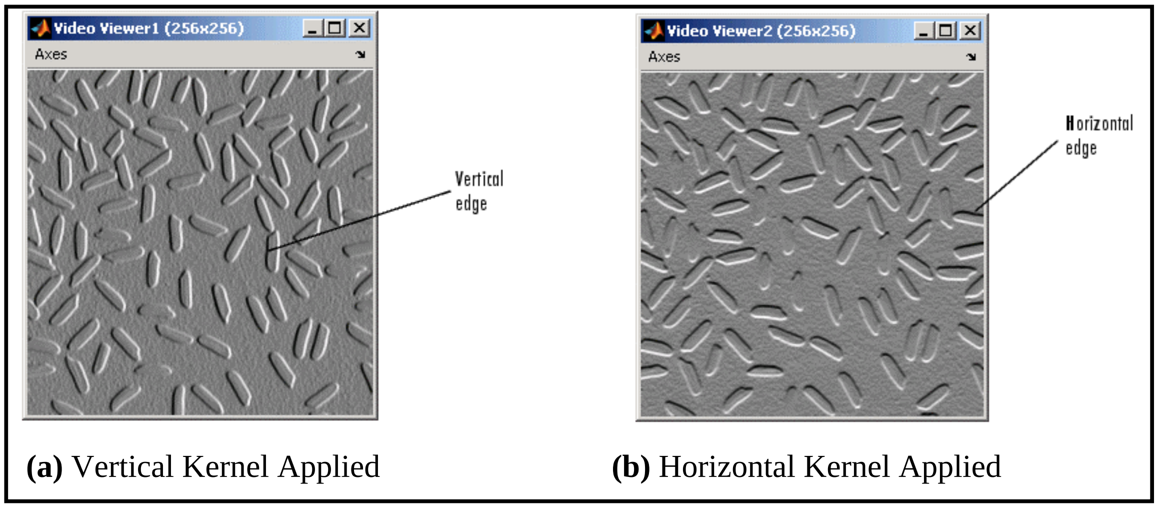


Figure 3.12: Visual Example of Sobel Edge Detection Kernels Applied Seperately[60]

Normally, the edge detection process would be complete with the calculation of the gradient magnitude, $|G|$, but Matlab® uses a slightly different approach to complete the edge detection process. Once it applies both kernels to the original image, the resultant gradient magnitude is found using Equation 3.21, above. Then, using a built-in threshold function, the background can be made black and the edges can be made white. This process essentially transforms the image into a binary image showing the edges only. This is illustrated below in Figure 3.13, with the final product of the edge detection process. This figure is the resultant image of the same grains of rice test image used above in Figure 3.12.

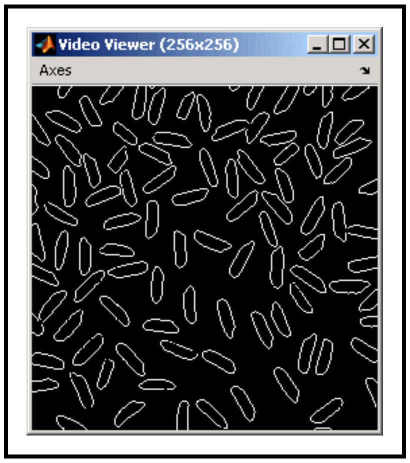


Figure 3.13: Visual Example of Sobel Edge Detection Final Product[60]

3.4.2 Morphological Filtering

Morphology consists of a broad set of image processing operations that filter images based on shape masks. These shape masks are called structuring elements; Section 3.4.2.1 describes the concept of a structuring element. Based on the definition of the structuring element certain shapes in an image can be emphasized or de-emphasized depending on what the interest is. This makes morphology a very useful filtering technique in operations where removing unwanted artifacts in the image is important in order to derive the correct information from the image. There are eight different morphological operations supported within Matlab®[60]. Of these eight, only three were used in this research effort and they will be described in Section 3.4.2.2 through Section 3.4.2.4.

3.4.2.1 Structuring Elements

An essential part of any morphological filtering operation is the structuring element which is used to probe the input image. Structuring elements consist of a two-dimensional matrix filled with zeros and ones and the structuring element is always much smaller than the image it is being applied to[64]. Just like a standard Cartesian coordinate system, the *origin* of a structuring element is the center pixel[64]. This location also identifies the pixel of interest when the structuring element is being applied to the input image. Furthermore, the pixels in the structuring element, which have the value of one, define the *neighborhood*[64].

Structuring elements can come in any shape desired. Most structuring elements are created using a premeditated shape because the shape of the structuring element defines the type of filter that a morphological operation becomes. Matlab® has some predefined structuring elements that reflect the most common elements used for noise filtering, feature extraction, and line detection. These elements are in the shape of a line, a disc, a diamond, or a square. The exact use of structuring elements will be described in detail in the forthcoming sections regarding each morphological operation. Examples of these structuring elements can be seen on the next page in Figure 3.14. Figure 3.14a is an example of a diamond shaped element, Figure 3.14b is an example of a line shaped element, while Figure 3.14c is an example of a square element.

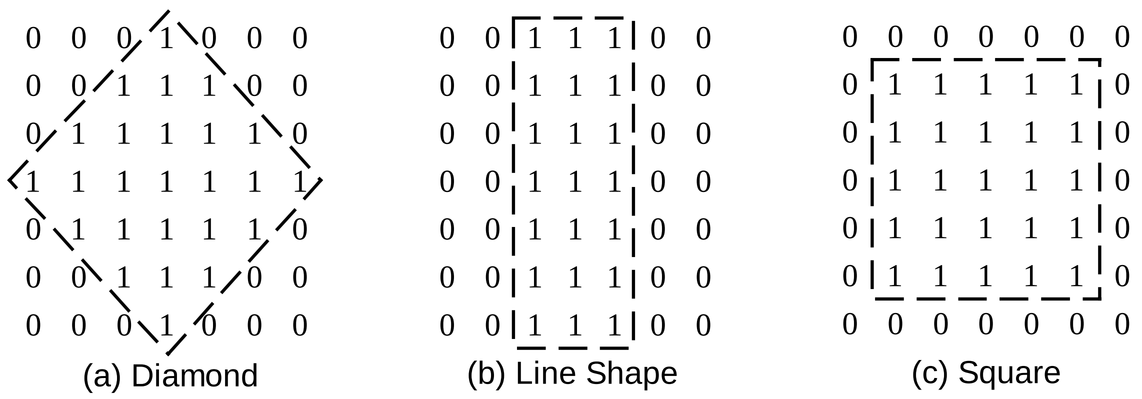


Figure 3.14: Example of Various Styles of Structuring Elements

3.4.2.2 Morphological Dilation

Morphological dilation is a process which allows a region of white pixels in an image to be able to grow in size. This may be desirable in order to fill in holes or to join two regions together in an image. The direction of growth can be adjusted by the design of the structuring element and this will be seen in examples below. Set theory is often used in MV in order to describe what the functions are actually doing to the image to perform its task. The definitions within Minkowski set theory will not be reviewed here, but set theory will be used to describe the individual functions. The definition of dilation, in terms of set theory, is as follows in Equation 3.22[64]:

$$Y = X \oplus B = \bigcup_{b \in B} X_b = \bigcup_{x \in X} B_x = B \oplus X \tag{3.22}$$

where Y is the set of pixels with a value of one that make up the output image, X_b is the set of pixels with a value of one that make up the original image, and B_x is the set of pixels that make up the structuring element neighborhood. The exact sequence for performing morphological dilation is as shown on the following page in Figure 3.15.

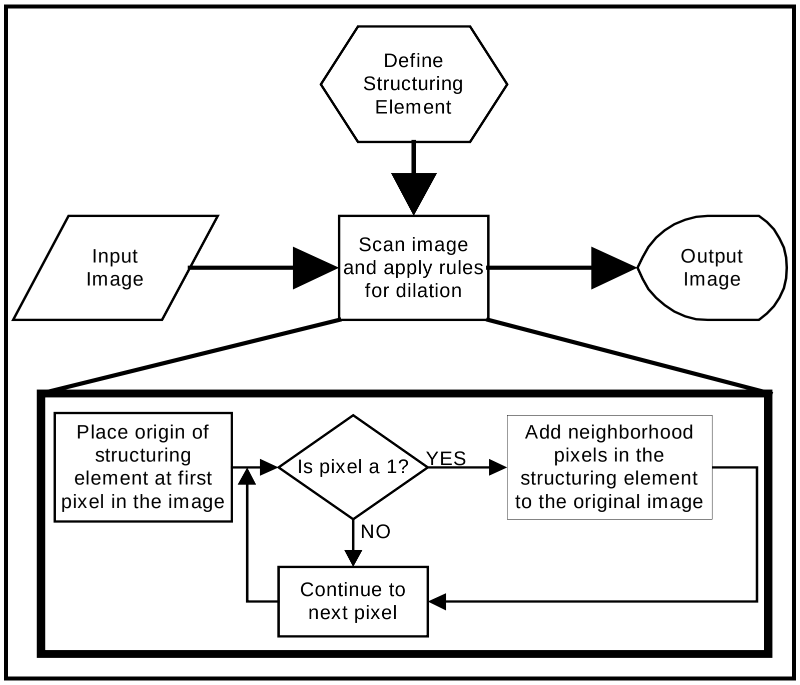

Define Structuring Element
Input Image
Scan image and apply rules for dilation
Output Image
Place origin of structuring element at first pixel in the image
Is pixel a 1?
YES
NO
Add neighborhood pixels in the structuring element to the original image
Continue to next pixel

Figure 3.15: Flowchart Indicating Process for Morphological Dilation

A visual example of morphological dilation will be presented in Section 3.4.2.4. along with an example of morphological erosion which, when combined, make up a morphological opening. Please refer to Section 3.4.2.4 for a visualization of the dilation process.

3.4.2.3 Morphological Erosion

Morphological erosion is very similar to morphological dilation. This process is meant to allow objects in an image to shrink. This is desirable where a feature is comprised of many layers of pixels, such as a line, and this feature needs to be detected by some other algorithm. This erosion can be used first to shrink the line down to one pixel thick, so that it is easier to detect as a line rather than an object such as a rectangle. This is used extensively in fingerprint matching algorithms where the lines from the fingerprint are shrunk to one pixel thick. As with the dilation, the direction of shrinkage can be adjusted by the design of the structuring element and this will be seen, again, in examples below. The definition of erosion in terms of set theory is as follows in Equation 3.23[64]:

$$Y = X \ominus B = \{x : B_x \subset X\} \tag{3.23}$$

where Y is the set of pixels with a value of one that make up the output image, X is the set of pixels with a value of one that make up the original image, and B_x is the set of pixels that make up the structuring element neighborhood. The exact sequence for performing morphological erosion is as follows in Figure 3.16.

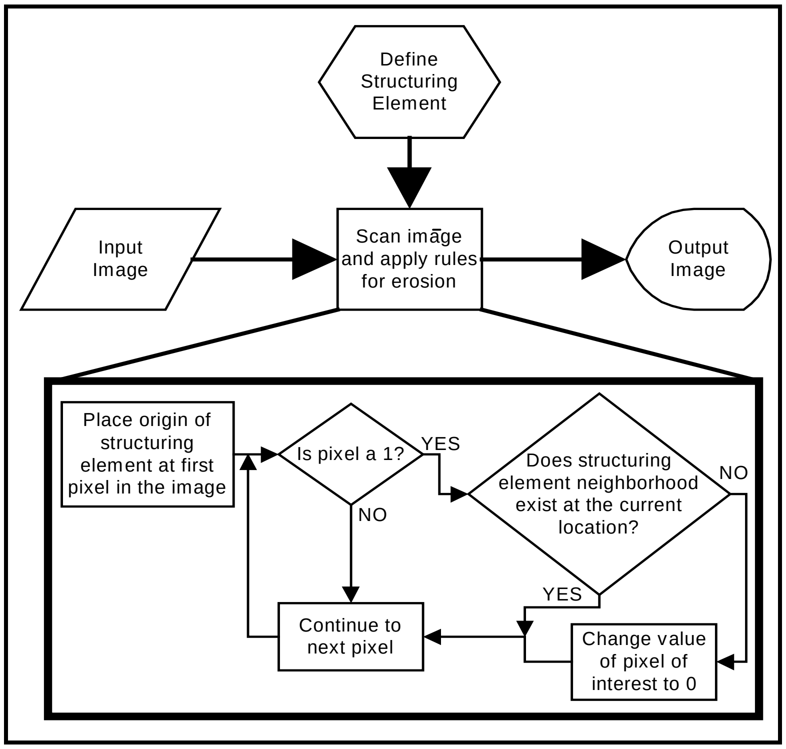

Define Structuring Element
Input Image
Scan image and apply rules for erosion
Output Image
Place origin of structuring element at first pixel in the image
Is pixel a 1?
YES
NO
Does structuring element neighborhood exist at the current location?
NO
YES
Continue to next pixel
Change value of pixel of interest to 0

Figure 3.16: Flowchart Indicating Process for Morphological Erosion

By looking at the flowchart, above, it can be seen that morphological erosion requires one more decision step than dilation does. This is fairly insignificant, but it should be noted because even so much as one more processing step can cause a slightly longer processing time, especially in larger images. As in the case of the morphological dilation, a visual example of morphological erosion will be presented in Section 3.4.2.4. along with the example of morphological dilation which, when combined, make up a morphological opening. Please refer to Section 3.4.2.4 for a visualization of the erosion process.

3.4.2.4 Morphological Opening

Morphological opening is a basic workhorse in image processing for noise removal and it can also be used to find certain shapes in the image that are defined by the structuring element. Basically, morphological opening consists of first performing an erosion followed by a dilation. The effect is that all of the stray pixels are removed by the erosion and then the object is regrown to resemble its original size and shape but without the 'outlier' noise pixels. The definition of opening as defined by set theory is as follows in Equation 3.24[64]:

$$Y = X \circ B = (X \ominus B) \oplus B \tag{3.24}$$

where Y is the set of pixels with a value of one that make up the output image, X is the set of pixels with a value of one that make up the original image, and B is the set of pixels that make up the structuring element neighborhood. Because opening is simply a combination of dilation and erosion, a flowchart will not be presented.

A slightly more advanced illustration depicting the use of opening for noise removal is presented on the next page in Figure 3.17. It can be seen in this figure that there are several outlying pixels that are considered to be noise. After the initial erosion is complete, almost all of the pixel information in the image has been lost. Because of the shape of the structuring element and the nature of the process of dilation, after the dilation is complete, all of the important pixel information in the image is regained and the noise has been removed. In this figure, the blue dots represent pixels that comprise objects who need to be separated from the noise in the image. The pixels that are considered to be noise are depicted as green dots. In this figure, the pixels that change state but are in the original image will have no color; this is because most of these pixels will come back in the second step. It should be noted that when performing a noise-filtering opening, the structuring element is almost always a 5×5 or a 9×9 matrix of ones.

In this simple example, a 3×3 structuring element is used. As stated before, this is a very effective tool for noise removal in binary images and was used throughout this research to 'clean up' the image sequence in the pre-processing stage prior to the application of any higher level MV algorithms.

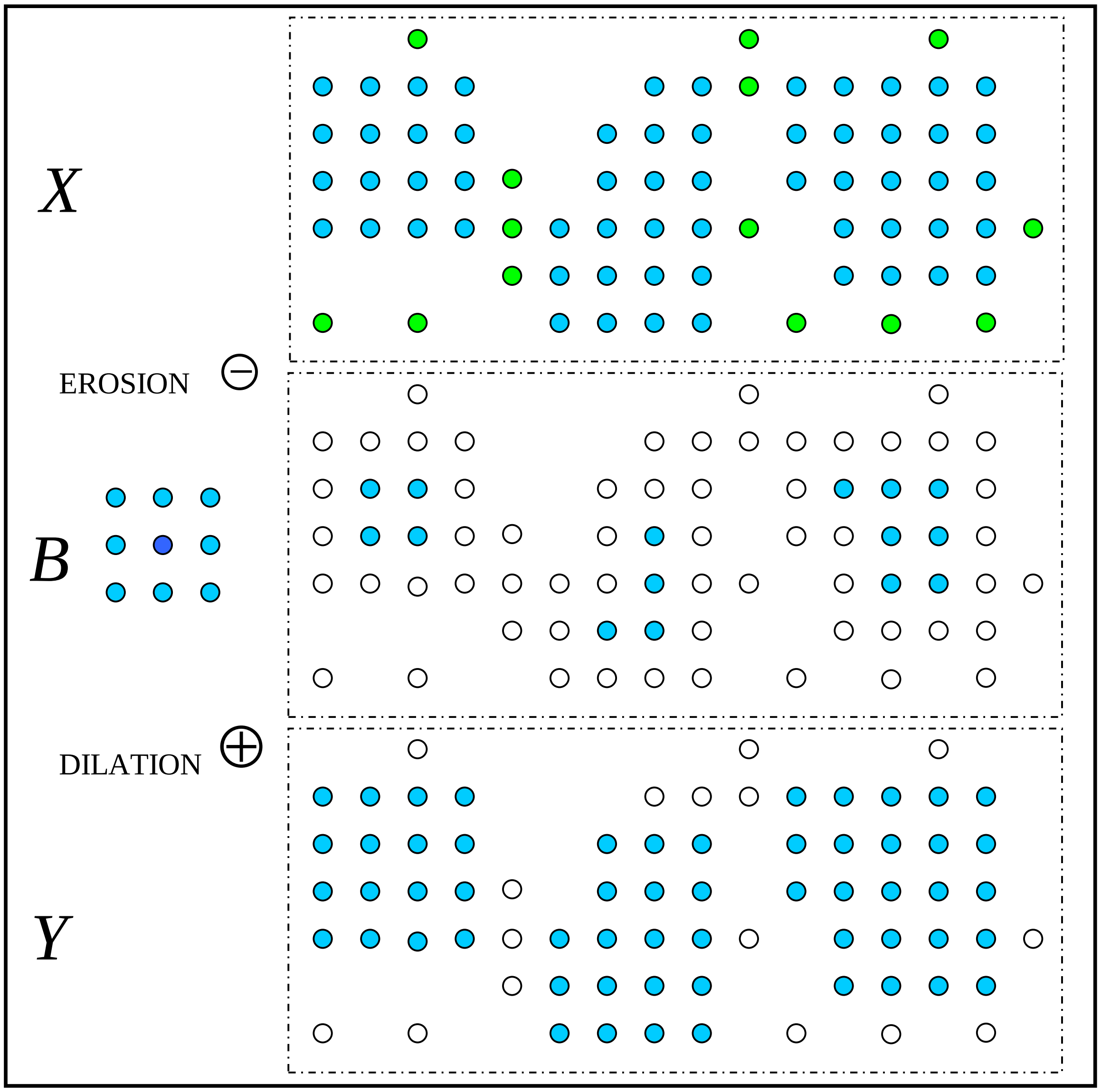


Figure 3.17: Visual Example of a Simple Morphological Opening

3.4.3 Line Detection Algorithms

There are two possible methods that immediately present themselves when thinking or researching about line detection. These methods are called edge detection and the Hough transform. Edge detection is a fairly simple routine that is easy to implement and this makes it a good candidate for such an application But, edge detection is a primitive routine and its ability to yield consistent and reliable results is almost non-existent. This is where the Hough transform steps in. The Hough transform is specifically a line-detecting algorithm. Therefore, it is thought that it must incorporate some ideas that will make its implementation and rate of success much higher than that of the edge detection. Both of these methods were probed for their feasibility of application to this problem of runway detection and the findings follow.

Edge detection is one of the most fundamental aspects of image processing that could be used in the runway detection process. If the image of the runway is looked at as edges or straight lines, the edges of the runways in particular stand out a great deal. This is a great point, but other things also stand out such as the horizon, roads, rivers, and bridges because these things also have edges or lines associated with them. These are the things that should be ignored when searching for the runway in the image. This is fairly tough since when an edge detection routine such as the Sobel algorithm, which was described earlier, is performed, the edges are not sorted out automatically. All of the edges are made equally as prominent and this makes edge detection very difficult to use by itself.

The Hough transform on the other hand is considered to be a more specialized, higher-level image processing algorithm. As such, it carries with it certain things that make it a fairly complex idea, yet easy to use and very effective. The Hough transform uses a mathematical transformation in combination with a search for global maximums in the output matrix to perform the line detection. After review of these two methods, it was decided not to attempt to rely on edge detection alone as the method for runway detection. It is felt that it would be better to use edge detection as a low level filtering technique and use the Hough transform as the main line detection algorithm. This is the approach taken in this solution.

3.4.3.1 Hough Transform

The Hough transform uses a mathematical transformation for detecting lines in an image. It is essentially a method for finding straight lines hidden in large amounts of data which is the same thing as line detection. The difference in the two methods is that with the Hough transform a certain number of lines can be detected based on their strength and a subsystem can be implemented to take care of false positives. This description makes it perfect for the purpose presented here regarding runway detection. The underlying principle of the Hough transform is that there are an infinite number of potential lines that pass through any point, each at a different orientation. The purpose of the transform is to determine which of these theoretical lines pass through the greatest number of features in an image.

In order to determine that two points lie on the same potential line, it is necessary to create a representation of a line that allows meaningful comparison and this is what the Hough transform does. In the standard Hough transform, each line in the original image is represented by two parameters called rho (ρ) and theta (θ), which represent the length and angle from the origin of a normal to the line in question. In other words, a line is described as being at an angle 90 degrees from θ, and being ρ units away from the origin at its closest point. See Figure 3.18, below, for an illustration of the description of how the Hough parameters relate to the original image space.

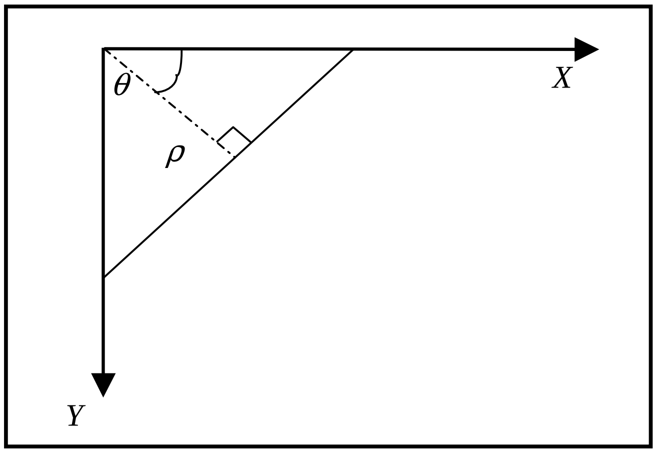


Figure 3.18: Relationship of Hough Parameters to Original Image Space

By transforming all of the possible lines through a point into this coordinate system, which means calculating the value of ρ for every possible value of θ, a sinusoidal curve is created which is unique to that point. This representation of the two parameters is referred to as the Hough space. If the curves corresponding to two points are superimposed, the locations in the Hough space where they cross correspond to lines in the original image that pass through both points. An example image of a straight line with some noise is shown in Figure 3.19, below. This image was used to perform a simple Hough transform for illustration purposes. The illustration of Hough space with points on the straight line in Figure 3.19 represented as sinusoids is found below, in Figure 3.20.

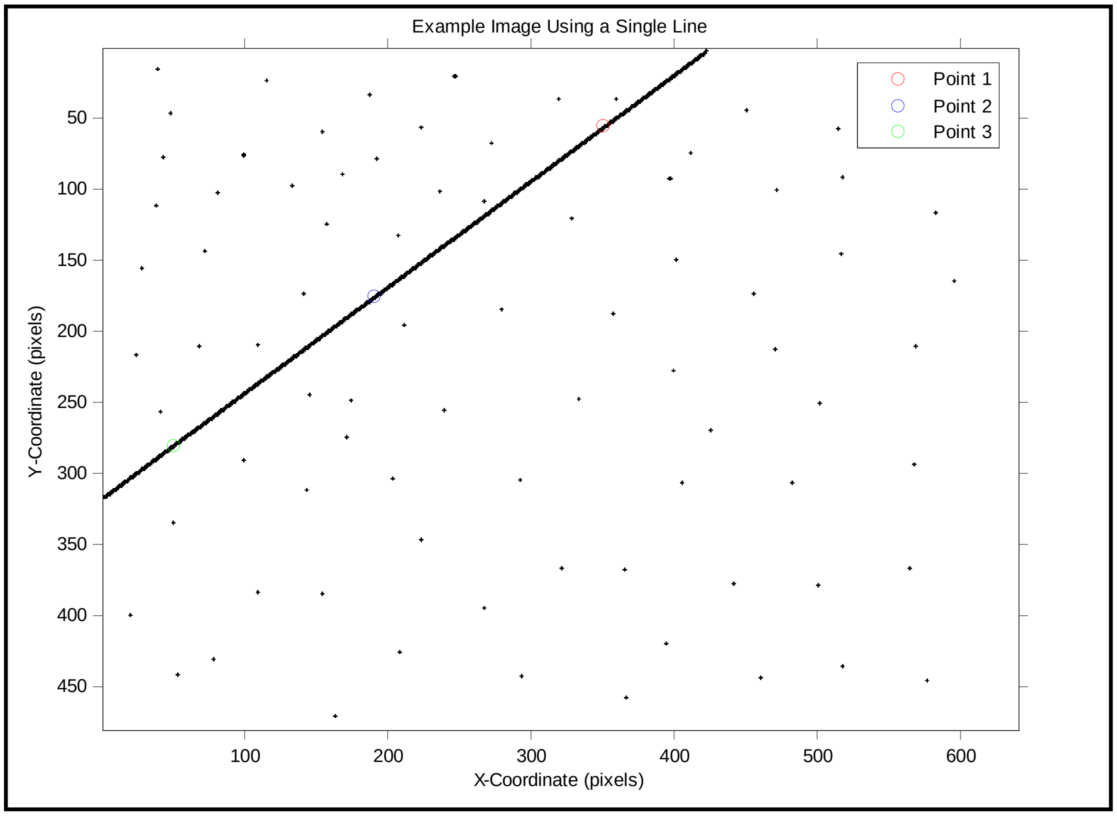


Figure 3.19: Example Image Used to Perform Simple Hough Transform

This figure is of a simple line drawn with a drawing program. The points used to calculate the Hough lines in Figure 3.20, on the next page, are shown by the red, blue, and green circles. These three points correspond to three sinusoids in Hough space in

Figure 3.20. The point in which these three sinusoids cross represent a single ρ and θ value. This is also called a Hough peak. In this example, the Hough peak would have a value or strength of three because three sinusoids are crossing at this point. If the inverse transform were to be applied, the endpoints of the line in the input image could easily be found and plotted. This is the exact sequence of operation of the Hough transform.

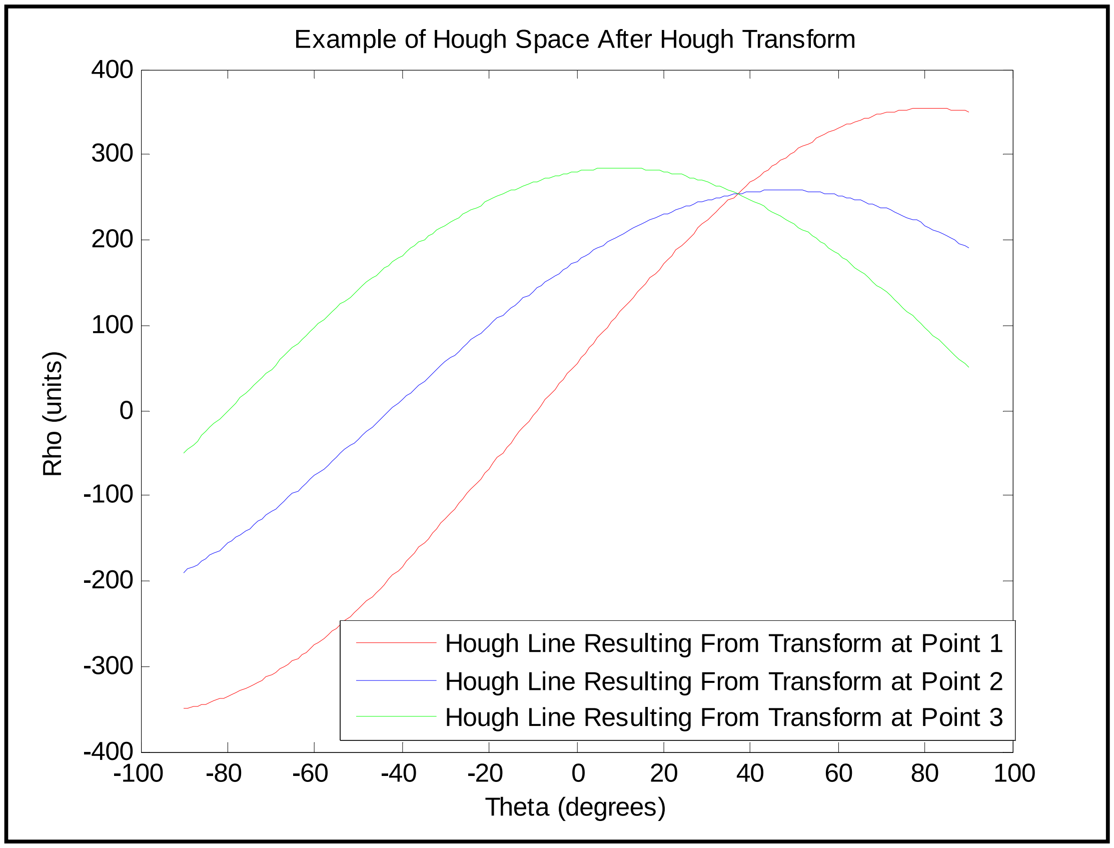


Figure 3.20: Hough Space Resulting From Hough Transform

The implementation of the Hough transform is not complex, mathematically speaking. In particular, the Hough transform is described by Equation 3.25[64], below:

$$\rho = x\cos(\theta) + y\sin(\theta) \qquad (3.25)$$

where x is the X-coordinate of the pixel of interest, y is the Y-coordinate of the pixel of interest, and θ is the range of values which are used in the calculation to get a corresponding list of ρ values.

Once a list of ρ and θ values are compiled for one single white pixel on the input image, the sinusoid is plotted in the Hough space. This process of calculating and plotting sinusoids is continued until sinusoids corresponding to the ρ values calculated from the corresponding discrete range of θ values are plotted in Hough space for every white pixel in the image frame. Once this is complete, a function, which determines maximums of values or strengths of peaks in the Hough space, is implemented, which scans the Hough space and will find the specified number of Hough peaks based on a threshold. Each peak in the Hough space corresponds to a place where many sinusoids cross at one point. This point represents a specific ρ and θ value, which inversely corresponds to a line in the original image. A flowchart highlighting this process of line detection is shown in Figure 3.21, below.

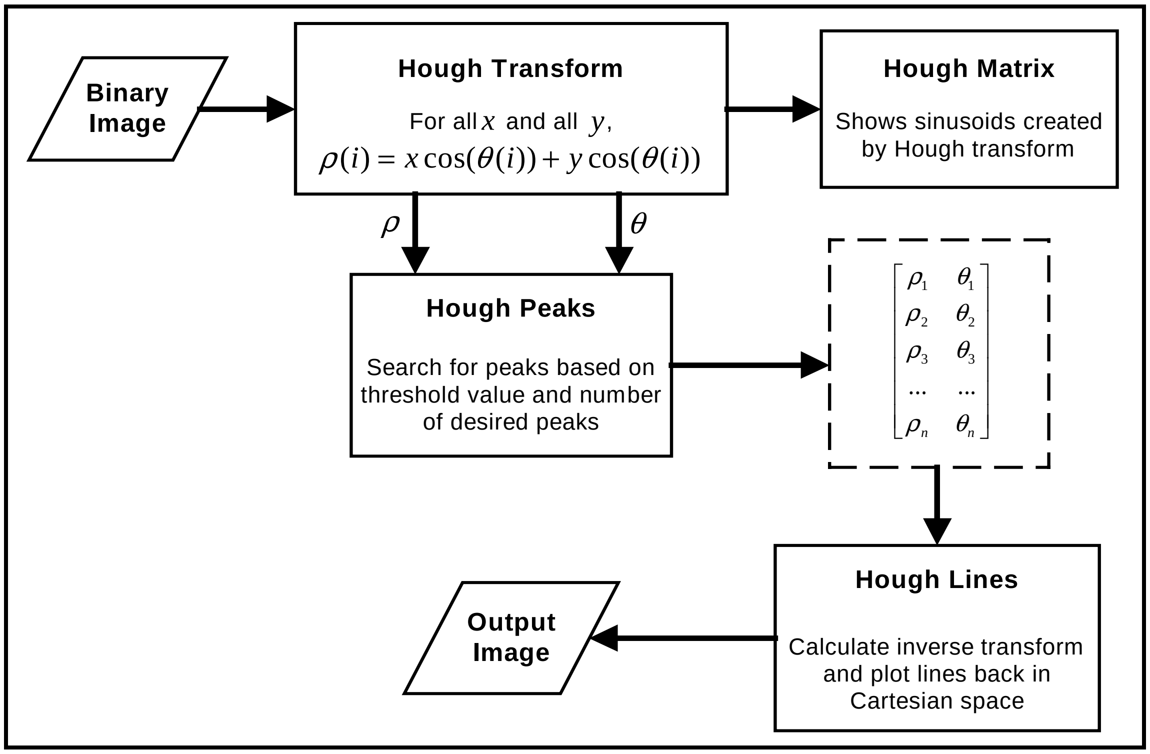

Binary Image
Hough Transform
For all x and all y,
$\rho(i) = x\cos(\theta(i)) + y\cos(\theta(i))$
Hough Matrix
Shows sinusoids created by Hough transform
ρ
θ
Hough Peaks
Search for peaks based on threshold value and number of desired peaks
$\begin{bmatrix} \rho_1 & \theta_1 \\ \rho_2 & \theta_2 \\ \rho_3 & \theta_3 \\ \cdots & \cdots \\ \rho_n & \theta_n \end{bmatrix}$
Hough Lines
Calculate inverse transform and plot lines back in Cartesian space
Output Image

Figure 3.21: Implementation of Hough Transform to Detect Straight Lines

Chapter 4

Experimental Procedures

4.1 Overview of Experimental Procedures

The experimental procedures necessary to address the problems required not only the development of software algorithms to accomplish the objectives outlined in Section 1.3 but also, the application of this software using real hardware in a laboratory setting. This provides an advantage in the level of assessment that can be attained from this research. By using real hardware and real images, many more issues are addressed than would be if a virtual image were generated and used. The experimental procedures for the two problems addressed in this research effort can be described in two sections. The first section, Section 4.2, includes the hardware and software used for the Marker Detection and Tracking problem. More specifically, Section 4.2 contains the hardware descriptions, the hardware setup, and the full description of the implementation of the Marker Detection and Tracking algorithms. The second section, Section 4.3, includes the hardware and software used for the Runway Detection problem. This section, like Section 4.2, also contains the hardware descriptions, and hardware setup, and the full description of the implementation of the Runway Detection scheme as well as an in depth description of the graphical used interface (GUI) used to control the simulation.

4.2 Experimental Procedures for the Marker Detection and Tracking Problem

The experimental procedures for this problem consist of a combination of hardware selection/setup and software setup. The merging of the software with the hardware is very much dependent upon the exact hardware setup that is and exactly what type of information can be gathered from the hardware outputs. This is an indication as to how much work the software will actually have to perform in order to accomplish the goals set forth for this problem. The overview of this blending of the hardware and software for the Marker Detection and Tracking problem can be seen in Figure 4.1, on the following page.

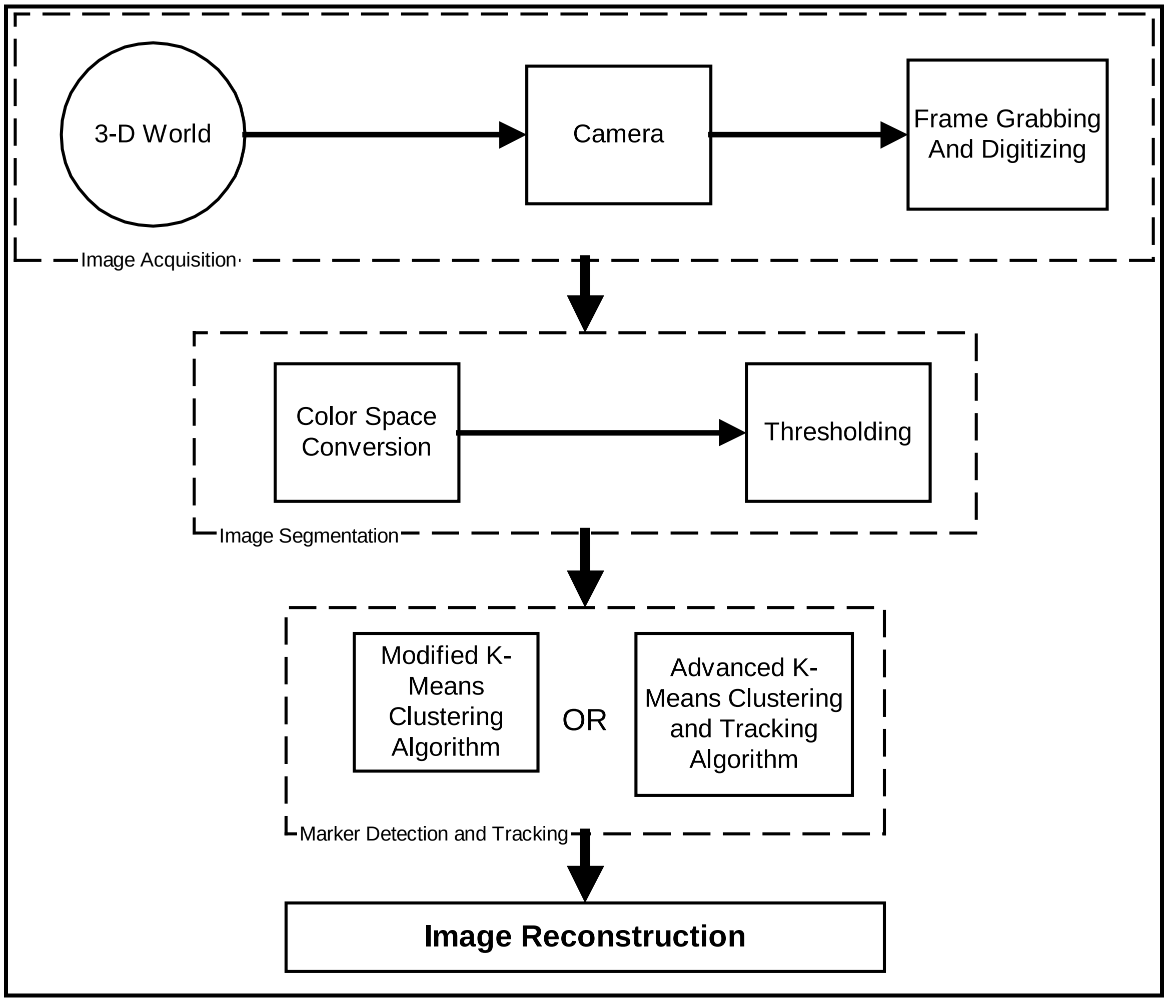

3-D World
Camera
Frame Grabbing And Digitizing
Image Acquisition
Color Space Conversion
Thresholding
Image Segmentation
Modified K-Means Clustering Algorithm
OR
Advanced K-Means Clustering and Tracking Algorithm
Marker Detection and Tracking
Image Reconstruction

Figure 4.1: Marker Detection and Tracking Experimental Procedures

4.2.1 Hardware Used for the Marker Detection and Tracking Problem

This section is dedicated to the physical and functional description of each piece of machine vision hardware used to address the Marker Detection and Tracking problem. Through this, the function, importance, and experimental procedures for each part as it relates to the problem will be outlined.

4.2.1.1 Description of Hitachi CCD Camera and Fujinon Lens

The camera used in the laboratory experiments for this research is a Hitachi brand, model KP-M22A. The KP-M22A is a compact, lightweight, black and white camera. The camera uses a high grade ½” charge coupled discharge (CCD) chip which produces a usable resolution of 768 by 494 pixels. The camera is powered by a +12 volt supply which is provided through the video bus cable from the frame grabber card. The parameters describing the cameras features are listed below in Table 4.1.

Table 4.1: Hitachi KP-M22A Specifications

Imaging Device	½" Interline CCD
No. Of Effective Pixels	768(H) x 494(V) pixels
Sync System	Internal/External (Automatically Switched)
Sensitivity Switching	FIX, AGC, or MANUAL
Gamma Correction	1 or correction
Electronic Shutter Modes	1/100 to 1/10,000
External Trigger	Field on Demand
Lens Mount	C-Mount
Power Supply	+12 VDC
Dimensions	29(H)×29(W)×62(D) mm
Mass	100 g

This camera has many features which include it in the list of high end industrial type machine vision cameras. All of these features are not needed for the purpose of this research but, the ability of the camera to have a high shutter speed, gamma correction, a ½" CCD and be small and lightweight were the determining factors in the purchase of this camera. For these experiments the use of the frame synchronization system was not used, nor was the external trigger options. The gamma correction was set to correction which applies a gamma correction of 2.4 and satisfies the Matlab® image standard of using a gamma corrected image. The shutter speed is dual in-line package (DIP) switch selectable and it was set to 1/100 of a second, which is acceptable for almost any conditions found in the lab environment. The sensitivity was set to FIX so as not to allow the camera to adjust the video gain to control the brightness. The reasoning behind this is that the brightness was to be controlled by the aperture of the lens which is easier to adjust and control by the user. The camera is shown on the following page in Figure 4.2.

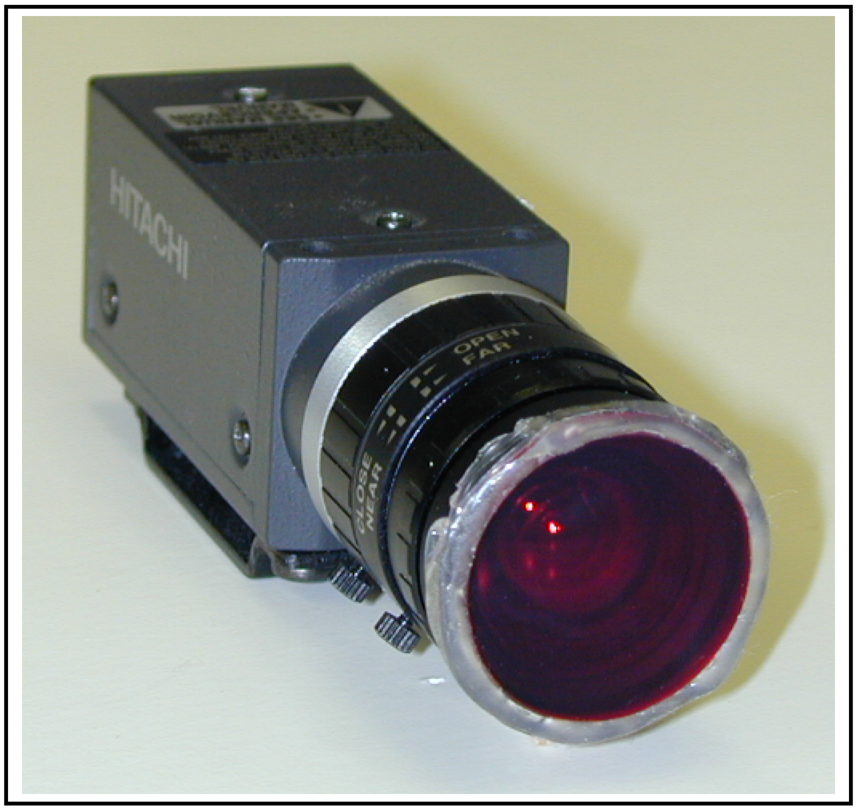


Figure 4.2: Hitachi KP-M22A Machine Vision Camera

The lens attached to the camera is a lens designed for general machine vision applications. The lens was selected so that the field of view would be approximately 4 ft. by 3 ft., which is appropriate for the type of simulations being conducted in a laboratory environment. The lens is a Fujinon brand, model DF6HA-1; it has a 6 mm focal length and it is designed for a camera which uses a ½” CCD. The full details of the Fujinon lens is found below in Table 4.2.

Table 4.2: Fujinon DF6HA-1 Specifications

Application	For ½” format CCD
Focal Length	6 mm
Focus Range	∞ - 0.1 m
Field Angle	56° Horizontal/44° Vertical/67° Diagonal
Field of View @ 1 meter	1.06 m (W)×0.79 m (H)
Iris Operation	Manual
Focus Operation	Manual
Mass	55 g

The color filter attached to the lens, shown in Figure 4.2, is a type of mechanical optical filter which is designed to enhance the red part of the visual spectrum of light. The basis for the use of this type of filter is to reduce the number of processing steps in software. By using this hardware type of filter, any red light that is viewed by the camera will show up as white to the black and white camera. This greatly intensifies the red markers in the image of the tanker and allows the software to take a more 'economical' approach in finding the markers.

4.2.1.2 Description of Euresys Picolo Frame Grabber PCI Card

The Picolo frame grabber peripheral component interface (PCI) card was selected for this research because of its outstanding price/quality ratio. The Picolo is a full featured frame grabber capable of capturing images in color or monochrome format in resolutions up to 768 by 576 pixels. The card can capture individual images as well as video sequences and write them to the computer's memory. This model of frame grabber card is designed to drastically simplify any task associated with machine vision. The Picolo is suitable for single camera operations but it supports three different input formats at a frame rate of up to 30 frames per second. The card also has four input/output (I/O) lines that can be used as hardware triggers for image acquisitions or for triggers for external hardware. The Euresys Picolo Frame Grabber PCI Card is shown on the following page in Figure 4.3.

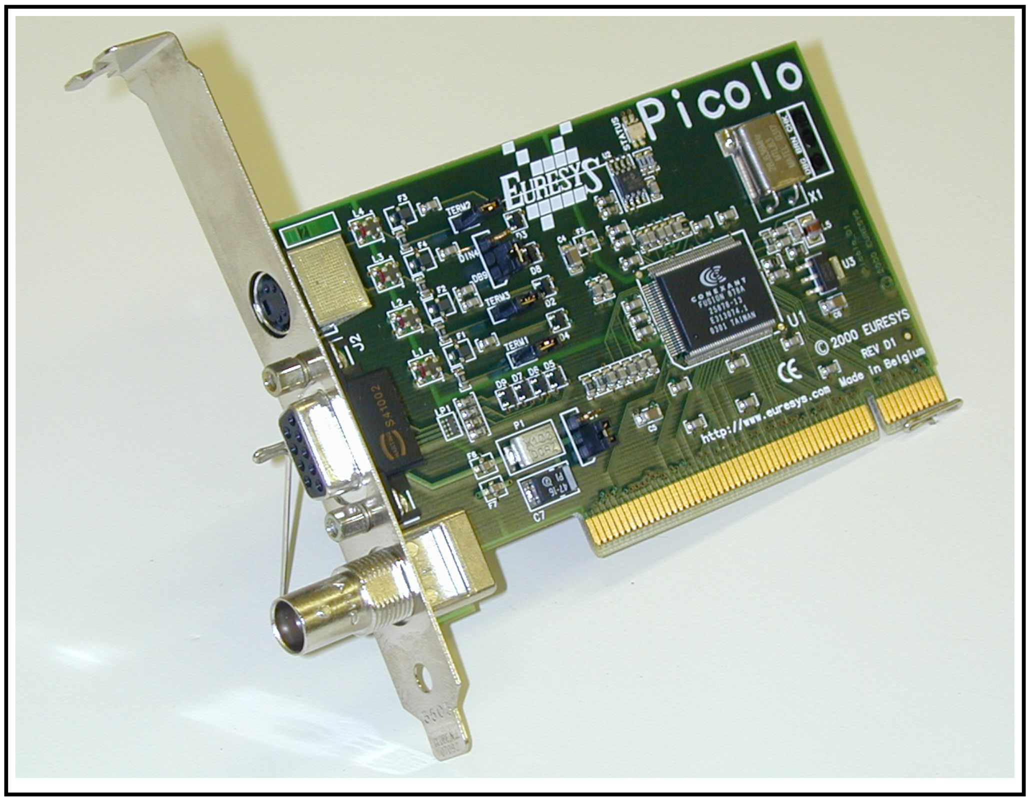


Figure 4.3: Euresys Picolo Frame Grabber PCI Card

4.2.1.3 Description of Machine Vision Research Computer

The MV research computer was purchased in pieces and assembled into the current machine. The purpose of buying separate pieces was to be able to buy the fastest components possible so that the computer would be well suited for MV research applications because it is widely known that MV applications take a great amount of processing power.

The computer was built using Intel framework utilizing a Pentium 4 class 3.20 gigahertz (GHz) Prescott processor seated in an Micro-Star International (MSI) brand, model 915G motherboard. This motherboard and processor combo allows the front side bus (FSB) to run at 800 megahertz (MHz) which was the fastest front side bus made at the time the machine was assembled. The speed of the FSB is an integral part of the speed of the computer because the FSB is the place where passing of information from memory to the main processor occurs. The machine is also using 512 megabytes (MB) of Double Data Rate 2 (DDR2) Synchronous Dynamic Random Access Memory (SDRAM) running at a speed of 2700 MHz. This is another very important part of the computer which must be fast to ensure the data transfer between internal parts is not bottlenecked in any way. The last part that must be fast is the hard disk drive. The hard disk drive in the machine is a special edition Western Digital 80 gigabyte (GB) hard drive running on a serial advanced technology attachment (SATA) bus. The SATA bus type of drive was selected because its speed in data transfer is superior to other previously used hard drive busses such as ATA 100 and ATA 133. This drive is a special edition drive because it has an enhanced seek time which further reduces the time it takes to store and retrieve data through the SATA bus.

The machine is also outfitted with a 17” flat screen monitor to save space. Its wireless mouse and keyboard enable the software to be manipulated while the user is standing next to the experiment. The machine vision computer system can be seen in Figure 4.4, below.

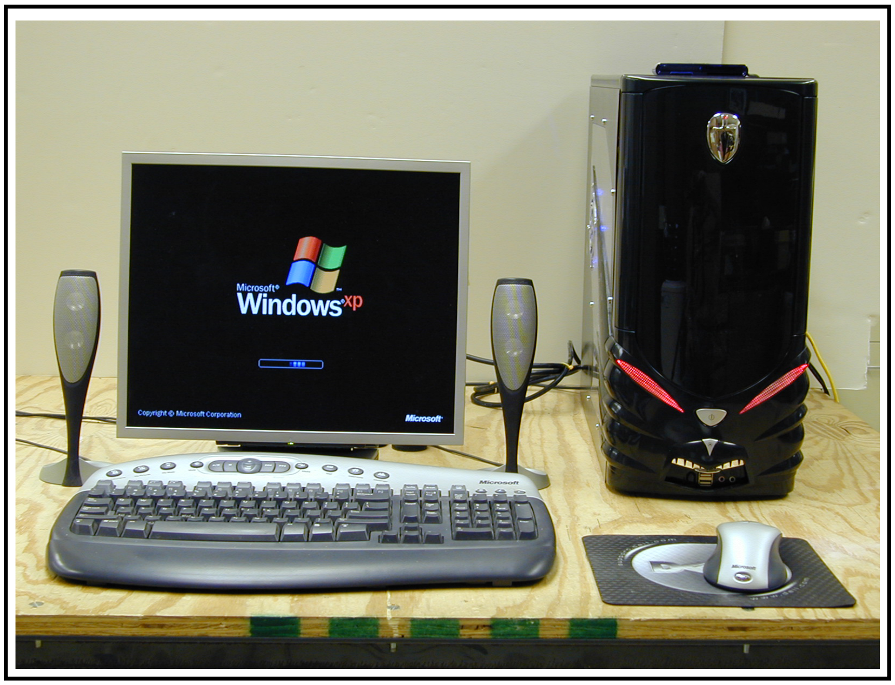


Figure 4.4: Machine Vision Research Computer

4.2.1.4 Model Aircraft and Camera Mount Apparatus

The model aircraft used in the laboratory simulations is a model of a Boeing 747 which is a typical tanker style aircraft. This aircraft was mounted into a sheet of blue foam board which was meant to emulate the color of the sky as a background. These aspects of the model aircraft and mount were attempts at achieving as much detail and to be as close to reality as possible in the laboratory environment.

The blue foam also has an axis hard mounted into the underside which allows it to be rotated about the longitudinal body axis of the aircraft. This allows tests and measurements to be performed on the part of the software which calculates the bank angle of the tanker using the marker positions. To validate the measurements, a large diameter compass was created and fixed to the table to allow visual angle measurements to verify the bank angle measurements given by the software. The aircraft's light emitting diode (LED) system is very simple, involving one resistor and a power distribution bus. The LEDs are powered by a single nine volt battery which must be wrapped in black tape to eliminate the glare off of its metal case from the overhead lights. The rotational axis is also removable such that the aircraft can be translated as well as rotated in order to evaluate the performance of other parts of the software. The aircraft, mount, LED system, and compass is seen on the following page in Figure 4.5.

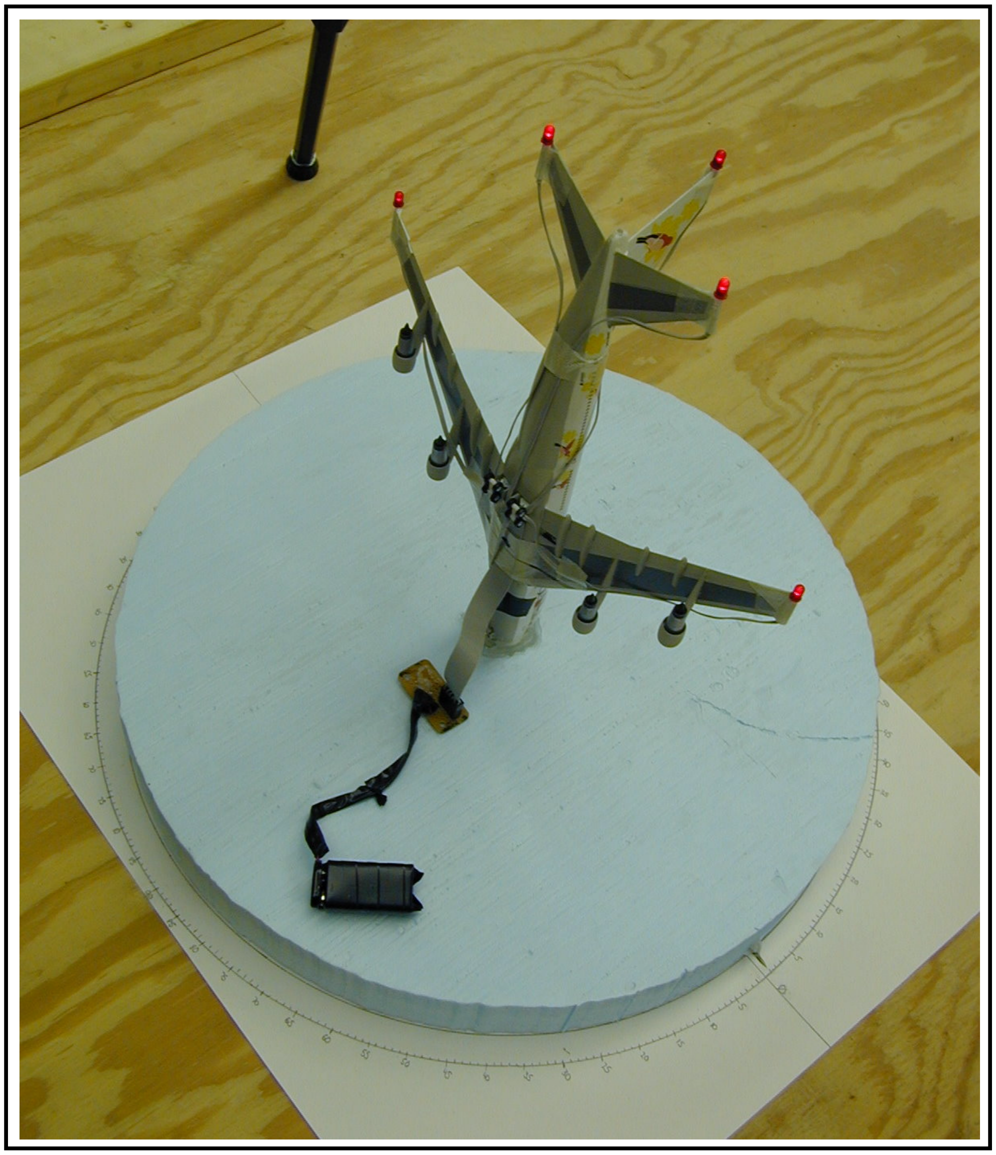


Figure 4.5: Model Tanker, Mount, LED System ,and Compass

The camera is mounted on a standard camera tripod which is hovering over the rear of the tanker model. This tripod is adjustable in height and the camera can adjust in many angles in order to ensure that the camera plane is parallel with the table, which is the most favorable position. The tripod and camera can be seen in Figure 4.6, on the following page.

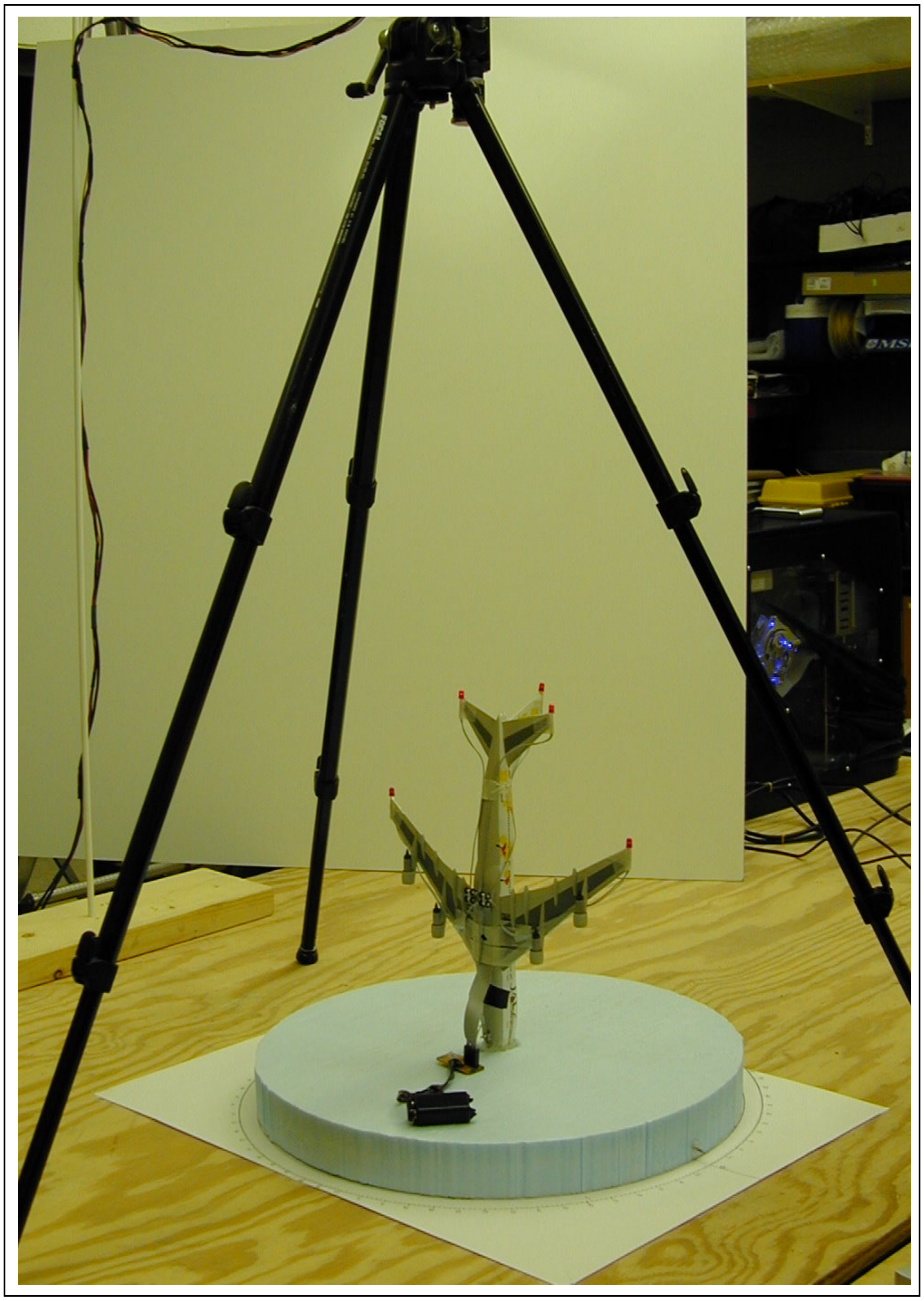


Figure 4.6: Tripod and Camera In Position Over The Model Tanker

Figure 4.7, on the following page, shows the typical view from above for the laboratory camera and model tanker equipment.

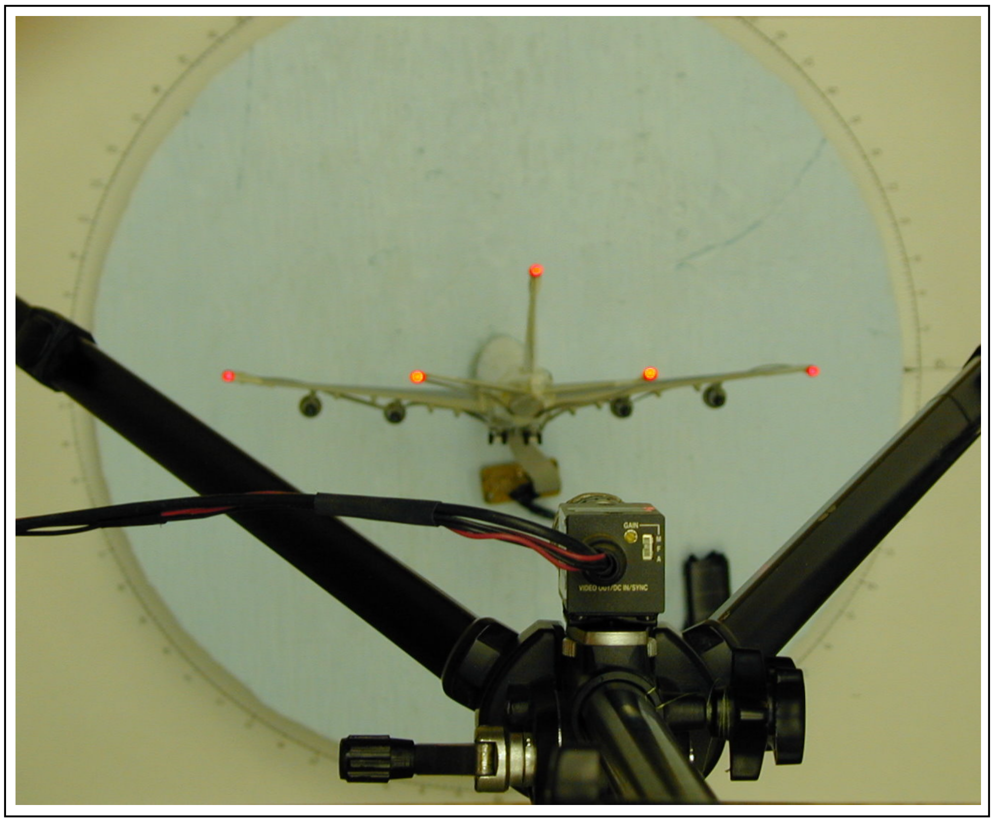


Figure 4.7: View From Above the Laboratory Camera and Model Aircraft

4.2.1.5 Camera Mount Noise Creation

In order to test the robustness of the software a source of noise was needed that could impact the stability of the camera such that the accuracy of measurements taken by the software was affected. It was determined that a good source of image noise would be vibrations. To impact the camera with vibrations, a source was needed. This source came in the form of a small electric motor. A small off-center weight was mounted on the motor such that when the motor was energized, a vibration was created. This motor was mounted to the top of the tripod, above the camera, such that the vibrations were intensified by the moment arm between the motor mount and the camera mount. The specifications of the motor are not known since it was a 'junkbox' motor but modeling of the noise was performed and it is described below..

To measure the vibrations, an inertial measurement unit (IMU) was employed. Since, the lab had ready supply of IMUs, the procurement of one for this purpose was not difficult. The IMU used is a Crossbow brand, model VG400. The VG400 was powered by an external power supply and it was mounted to the camera tripod as close to the camera as possible in order to attempt to accurately measure the vibrations that the camera was encountering. The VG400 was connected to the machine vision research computers serial port and the supplied software was used to record the accelerations felt by the IMU and camera. These accelerations were later used to quantify the vibrations and will be covered in more detail in Chapter 5. Figure 4.8, below shows a picture of the vibration motor attached to the tripod and Figure 4.9, below shows a picture of the IMU attached to the camera tripod.

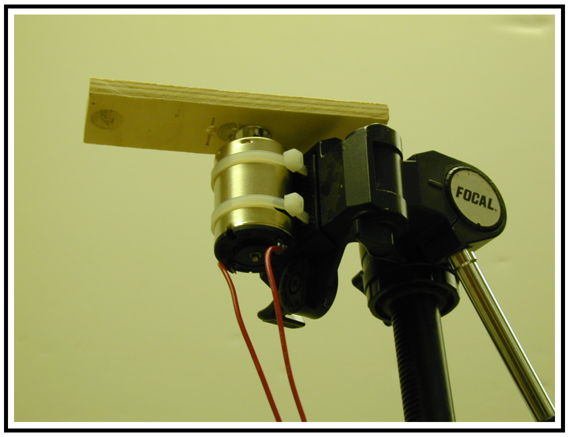


Figure 4.8: Vibration Motor Attached to Tripod

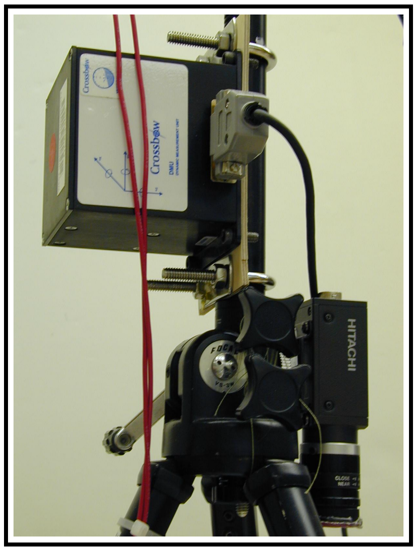


Figure 4.9: Crossbow IMU Mounted With Camera On Tripod

4.2.1.6 Limitations of the Marker Detection Hardware Setup

Before the experimental procedures are discussed further, the limitations of this setup should be discussed. Due to the fact that these experiments were performed in a laboratory environment which was fairly controlled, there are some issues that were not fully explored due to these limitations. The limitations are listed below.

1. *Lighting conditions were controlled* – Because the lighting conditions were controlled, an almost perfect depiction of the markers was available all of the time and this is certainly not indicative of the conditions experienced in a real situation. Possible solutions to this are to have the experiment inside an area where the lighting could be randomly generated such that the brightness of the lights varies independently of anything else. This would give a more realistic effect to this limitation.

2. *Scale of aircraft model with respect to the size of the markers* – The scale of the aircraft w.r.t the size of the markers was certainly not proportional. The availability of LEDs that would be size appropriate for the scale of the tanker were not readily available, therefore, the LEDs that were available were used and thus created an unfairness in that they are larger than could be expected in a real situation. This limitation enabled the software to detect the markers more easily than would probably occur in a real situation. This limitation could be rectified by obtaining information relating to the size of markers on a real tanker and scaling them appropriately.

3. *Distance of camera to tanker* – The distance of the camera to the tanker was also not proportional. This limitation also allowed the markers to be more easily detected than would normally be expected in a real situation due to the increased size of the tanker in the image frame. This limitation could be eliminated by either using a taller tripod or using a smaller camera.

4. *Tanker had limited motion* – The tanker was not able to be mounted on any kind of motion actuation system and therefore was left to be moved by hand in order to simulate the motion that could be encountered in flight. This simulated motion was certainly not what could be expected in a real situation due to the motion being much greater. Since the motion was much greater, the tests were unfair to the software in that they presented much greater motion than would normally be encountered. This limitation could be removed by attaching the tanker to an motion actuation system that would allow the tanker to move like a real aircraft in flight.

5. Tanker had a limited number of markers - Since the LEDs were much bigger than the tanker in scale, the number of markers was limited by simply not having enough area on the tanker to place more markers. This limitation presents an easier problem to the MV system than would normally be encountered in a typical AAR situation. This problem could be solved by simply increasing the size of the tanker model or decreasing the size of the LEDs.

These limitations have a direct effect on the real life performance of this algorithm. Therefore, these issues should be addressed before the results presented in this research are used for determining real life applicability of such an algorithm.

4.2.2 Software Used for the Marker Detection and Tracking Problem

The software used to address the Marker Detection and Tracking problem consists of two different methods, both of which accomplish the same result. The first method, whose theory was described in Section 3.3.1, is the Modified K-Means Clustering Algorithm. The second method, whose theory was described in Section 3.3.2, is the Advanced Modified K-Means Clustering Algorithm.

4.2.2.1 Modified K-Means Clustering Algorithm

In order to address the problem of Marker Detection and Tracking, the assumptions on which to base the software framework using the Modified K-Means Clustering Algorithm must be determined. In order to make this algorithm robust and have the ability to be used in a fairly uncontrolled environment, the assumptions must not be strictly confined. Keeping this in mind, the following list of assumptions was assembled for this problem.

<u>Assumptions for the Modified K-Means Clustering Algorithm:</u>

1. The number of markers is always fixed,
2. The wing tip markers must have the greatest distance to each other, the horizontal stabilizer markers must have the next greatest distance to each other,
3. The bank angle of the aircraft in question can never exceed 85 degrees.

Most of these assumptions fall into the ‘more than acceptable’ category in a real world environment, except for Assumption #1. This assumption is not favorable in a real world environment due to weather conditions or other factors that may exist that could obscure one or more markers. Although this assumption is a tough one to guarantee, it is required by the software because of the use of the K-Means Clustering Algorithm. The number of objects being searched for is the only constraint that must be fixed in the K-Means Clustering Algorithm. Therefore, for this research, this constraint must be applied.

This algorithm performs four basic image processing functions: Image Acquisition, Image Segmentation, Pixel Grouping, and Marker Labeling. The Image Acquisition is very straightforward and consists of simply grabbing an input image and digitizing it. Image Segmentation refers to, of course, discretely segmenting the image into parts that the software can discern useful information from. The Pixel Grouping function refers to physically constructing the desired number of groups of pixels, each representing one marker, from the global list of white pixels found in the image. The grouping function then calculates the centroids of each group and designates each marker's location as the location of the centroid. The Marker Labeling function gives each marker centroid location a name associated with the correct location on the aircraft. This labeling is necessary in order to tell if a certain group of pixels belongs to say, the left horizontal stabilizer tip or the vertical stabilizer tip. This is an essential operation if the algorithm is to be used with a pose estimation algorithm.

The first section of the software performs the image acquisition. A typical input image of the tanker aircraft with LED markers illuminated is shown in Figure 4.10, below.

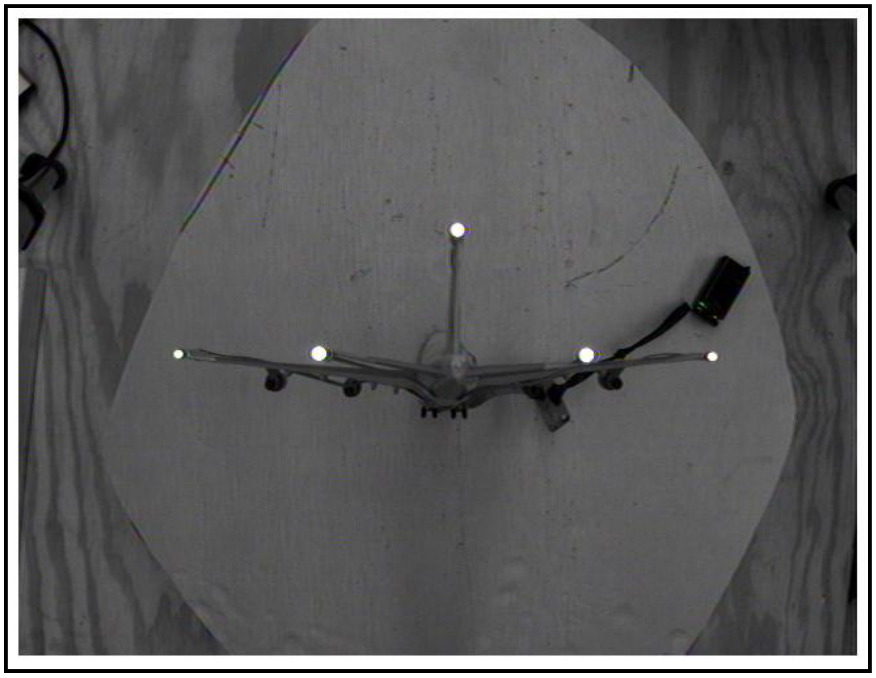


Figure 4.10: Typical Input Image of the Tanker Aircraft with LEDs Illuminated

Once the image acquisition is complete, it is followed by the Image Segmentation function, which, in itself, consists of two parts: Color Space Conversion and Thresholding. These two parts are described in detail in the following paragraph.

The first part of Image Segmentation is the color space conversion. A binary color space conversion is performed on the input image which converts it from a

640×480×3 grayscale image (pixel values range from 0 to 255) to a 640×480×1 binary image (pixel values range from 0 to 1). This operation was explained in detail in Section 3.2.3.2. Once this is complete, a thresholding operation is performed to accomplish basic image segmentation. This thresholding is designed such that all of the background pixels will change to a value of zero or black and all of the pixels representing the light markers will change to a value of one or white. This principle was explained in detail in Section 3.2.5. Once the thresholding is complete, the image is left in a state where all five markers are clearly defined by small groups of white pixels surrounded by an all black background. The Image Segmentation part of this algorithm is thus complete. An example of the image after the segmentation portion of the software is complete is seen below, in Figure 4.11.

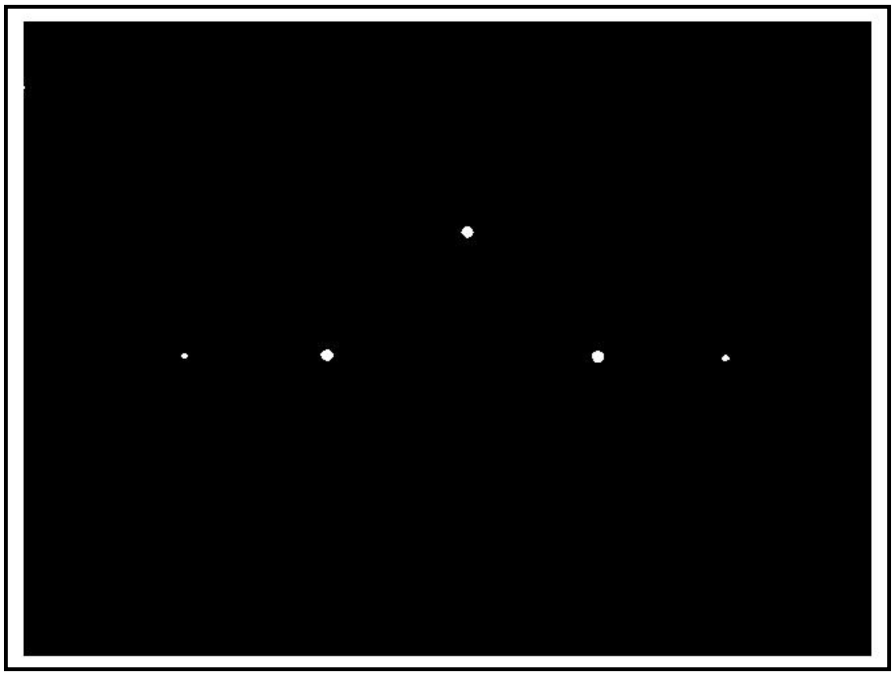


Figure 4.11: Mid-Stream Image After Performing Image Segmentation

The second main part of the algorithm, the Pixel Grouping section, is now ready to be performed. The pixels are grouped by first examining the X-coordinate and based on a distance threshold, the pixels are separated into distinct groups. Once the pixels are grouped by X-coordinates, then the Y-coordinate is examined. This can sometimes result in a more detailed grouping. This only occurs when the aircraft is at a bank angle which allows two markers to line up vertically. If it were not for this condition, the software could rely on the X-coordinate based grouping alone. In order to visualize how this problem can occur, refer to the list of white pixels is shown below in Table 4.3.

Table 4.3: Common Example of List of White Pixels Obtained

97, 350	**182, 91**
97, 351	**183, 251**
180, 250	**183, 252**
180, 251	**183, 90**
181, 250	**183, 91**
181, 251	**184, 252**
182, 250	**184, 253**
182, 251	**184, 91**
182, 252	**184, 92**
182, 90	**…**

In Table 4.3, a representative partial list of white pixels is shown and the colors represent the actual clusters that each pixel belongs to. The magenta pixels are the ‘Marker One’ pixels, the blue pixels are the ‘Marker Two’ pixels, and the green pixels are the ‘Marker Three’ pixels. If the grouping is based solely on the X-coordinate, it can be seen in this table how the pixels could be confused in their respective groups. In this example, all of the pixels in blue and green would have been grouped as one marker cluster, but by looking at the Y-coordinates it is easy to see that there is a large void between the two groups of Y-coordinates. The examination of the Y-coordinate allows the more detailed grouping in this case, thus creating three groups and not two.

Once the Pixel Grouping section is complete, the Marker Labeling section of the algorithm is performed. This is the last step in the Modified K-Means Clustering Algorithm. The Marker Labeling section of the algorithm is based upon a simple assumption about most aircraft, Assumption #2 in the list of assumptions. This assumption basically states that in most aircraft, the wings are always the longest ‘extensions’ from the centerline of the aircraft and they are always longer than the horizontal stabilizer. This is especially true for tanker style aircraft due to their long wingspan and this assumption must not be violated to ensure proper labeling is taking place. This will not be a problem because the aircraft is very unlikely to change its configuration in flight in such a way to violate this assumption, especially since tankers are non-reconfigurable aircraft. Even so, if an aircraft design is presented in which this assumption is broken, the software can easily be adjusted to accommodate the configuration of the new aircraft to ensure proper labeling of the markers. Assumption #3 is also very important in this labeling process. If this assumption is violated, the markers may be labeled wrong from the start or may become labeled wrong. The exact surfaces they represent will not be affected but the fact that they are on the left or right side of the aircraft will be affected. This will be explained in more detail in the following paragraphs.

Using these two assumptions, the markers are labeled by calculating the absolute distance combinations for all five markers. This means the distance from ‘Marker One’ to the other four markers will be calculated and so on, until all of the combinations have been calculated. These combinations consist of 10 different distance calculations. Along with these calculations, the marker numbers being used in the calculation are stored with each distance. In order to find the wing, the largest distance is found by using the maximum function within Matlab®. Once this value is found, the list of 10 distances is scanned for this one particular distance. When it is found, the marker numbers associated with that particular distance calculation are retrieved and the two marker numbers and their positions are known. The software can now positively say that those two markers belong to the wing tips. Once the wing tips are identified, the algorithm removes all of the distance calculations from the list that involved the two now identified wing tip markers, reducing the list to only three distances instead of 10. Then, the same method is used to find the horizontal stabilizer markers, using the maximum distance found in the now updated list. Once the horizontal stabilizer markers are identified, there is only one marker left and it is thus identified as the vertical stabilizer marker.

The software will distinguish between left and right hand side markers. This is accomplished two ways: one way has to do with how the image is scanned left to right, always encountering the left markers first, the second way is the use of Assumption #3.

It cannot be positively stated that the aircraft in question will never exceed 85 degrees of bank angle, but if this occurs, there are many more important issues to worry about than trying to approach or follow another aircraft. In Figure 4.12, below, the aircraft is shown in a radical attitude of approximately 85 degrees of bank angle and the markers are still being labeled correctly. Note that the method of labeling the markers allows the left and right sides to be distinguished while also labeling the wingtips, horizontal stabilizer, and the vertical stabilizer. The red markers are used to indicate the right side, the green markers are used to indicate the left side, and the blue marker is for the center. The marker shapes represent the different locations on the aircraft thus, the wing tips are represented by stars, the horizontal stabilizer tips are represented by circles, and the vertical stabilizer is represented by a diamond. The Matlab code for this algorithm can be found in Appendix A.

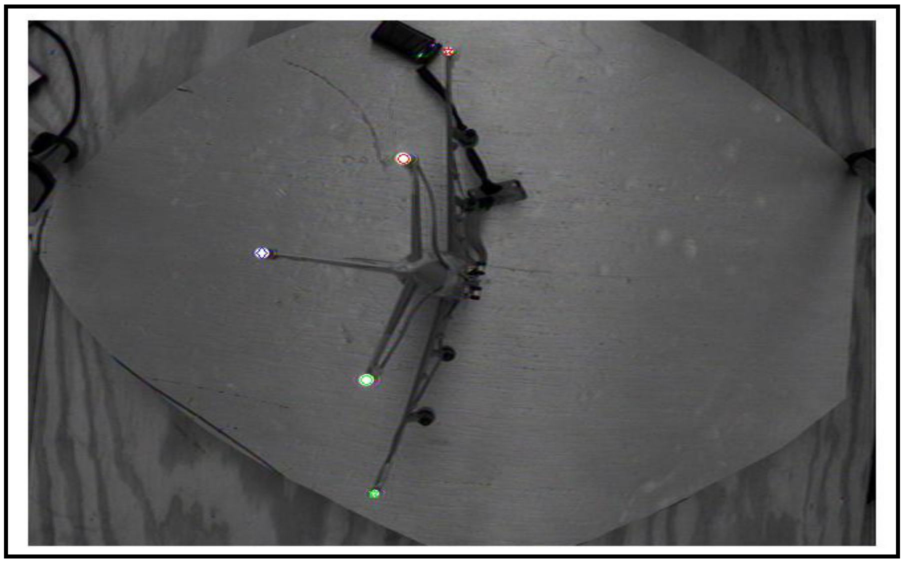


Figure 4.12: Typical Output Image with Aircraft Roll Angle ≈ 85º

<u>Possibility of Loss of Marker Visibility</u>

Upon review, a point was made that the possibility exists for one or more of the markers on the tanker to become obscured by the refueling boom during the refueling process. A marker may also disappear due to being damaged or burnt out. It was previously mentioned that when using the K-Means algorithm, the only thing that needed to be set was the number of clusters being searched for. This obviously presents a problem during the time when a marker would be obscured. Therefore, some changes were made to the Modified K-Means Clustering Algorithm to make it robust to this problem.

In the previous version of this software, the code for-looped through the marker detection section, one marker at a time up to the number of desired markers. If a marker did not exist, the code would halt due to the lack of a marker in the image frame. In this version of the software the number of clusters to be found are not set and a while loop is employed which will find any number of clusters instead of a set number. This has proven to be an effective way to deal with any number of markers including the loss or gain of them.

The use of this method of finding the clusters did present other problems. In the previous version of the software, the labeling of the markers was performed by calculating the distance between every possible pair of markers and determining which set of markers belong to the wingtips first, the stabilizer tips secondly, and the rudder was last. This order was chosen because the array of calculated distances could be searched for the maximum distance first, which should be attributed to the wingtips and then those distances which included the now defined wingtip markers would be removed from the array and the list would be searched again. On this subsequent search, the stabilizer tips would be found because they would now be the largest distance in the list. This continued until there was only one marker left in the list and that would be defined as the rudder. This scheme of labeling will not work if a loss or gain of marker situation is presented. Due to the fact that the marker could present itself anyplace in the frame due to the bank angle that the tanker could achieve, there is no way to label the markers using the hierarchical method that was used previously. A more advanced points matching and labeling algorithm[50,51,52,53,54] would have to be used in order to label the markers accurately. The addition of such a labeling algorithm was not within the scope of this research effort and thus was not attempted. This method, when compared to the previous method, is very much in contrast in that it does not require any assumptions. With the implementation of the while loop the previous Assumption #1 is no longer needed and since there is no labeling algorithm employed, then Assumption #2 and Assumption #3 is not needed as well.

In order to test this software, the model aircraft apparatus had to be modified to allow an additional LED and a switch that could activate and deactivate the LED at will. This type of setup was used to record two additional videos in which the tanker was in motion with the LED disappearing and reappearing. These videos were used to evaluate the computational workload of the software and to visually validate that it was working properly. These results will be shown in Section 5.1. The Matlab code for this more robust algorithm can be found in Appendix B.

4.2.2.2 Advanced K-Means Clustering and Tracking Algorithm

Once the Modified K-Means Clustering Algorithm was created, the Advanced K-Means Clustering and Tracking Algorithm was simple to implement. Its implementation consisted of adding a separate set of instructions to the initial piece of code that could calculate the velocity and accelerations of the markers that were found from the last three frames of video. Before these instructions could be solidified, the assumptions governing the software must be determined. The assumptions governing the Advanced K-Means Clustering and Tracking Algorithm follow.

<u>Assumptions for the Advanced K-Means Clustering and Tracking Algorithm:</u>

1. The number of markers is always fixed,
2. The wing tip markers must have the greatest distance to each other, the horizontal stabilizer markers must have the next greatest distance to each other,
3. For the initial conditions, the bank angle of the aircraft in question can not be greater than 85 degrees.

In examining the assumptions outlined above, the only difference between the Modified K-Means Clustering Algorithm and this algorithm is that the aircraft in question can now exceed 90 degrees of bank angle but not initially. This is a direct result of the implementation of the tracking part of the algorithm and this will be explained in detail in the following paragraphs. In order to implement the additional code for this version of software, the original piece of code was changed to run for only three time steps, after which the whole scanning of the image was eliminated and only small areas around the estimated positions were scanned for white pixels. This software was also written, first, to only use the marker velocities and no acceleration calculations at all, thus utilizing Equation 4.1:

$$\begin{bmatrix} x(index+1) \\ y(index+1) \end{bmatrix} = \begin{bmatrix} x(index) \\ y(index) \end{bmatrix} + \begin{bmatrix} V_x(index) \\ V_y(index) \end{bmatrix} \Delta t \tag{4.1}$$

where x is the X-coordinate of the centroids of the markers and y is the Y-coordinate of the centroids of the markers at frame number $index$. Δt is the time in seconds between the two sequential frames. V_x and V_y are the velocities in the x and y directions, respectively. This was implemented to give a middle baseline for comparison of the tracking improvement from using no inertial information (like in the Modified K-Means Clustering Algorithm), to using only velocity and, finally, to using both velocities and accelerations. These situations will be evaluated in Chapter 5.

Once the position is estimated using either Equation 4.1 or 4.2, below, the software will calculate a range of X and Y coordinates, creating a processing window, relating to each marker.

$$\begin{bmatrix} x(index+1) \\ y(index+1) \end{bmatrix} = \begin{bmatrix} x(index) \\ y(index) \end{bmatrix} + \begin{bmatrix} V_x(index) \\ V_y(index) \end{bmatrix} \Delta t + \tfrac{1}{2} \begin{bmatrix} A_x(index) \\ A_y(index) \end{bmatrix} \Delta t^2 \tag{4.2}$$

The concept of the processing window is simple and it is the heart of the expected increase in computational efficiency in this version of the software. This concept relates to the scanning of the image for white pixels. In the previous version of the software, the entire image was scanned and searched for white pixels. In this version, the entire image is scanned only three times, during the initial three frames of video. Once this is finished, the marker positions are estimated and then only a small area around the estimated position, based on a fixed square search area, is scanned on the subsequent image. This reduces the number of pixels to be scanned from about 300,000 in a 640 pixel ×480 pixel resolution image to about 500 pixels using a search area size of 10×10. This relates to scanning only 0.16% of the image compared to previously scanning 100% of the image.

Furthermore, by scanning only the areas around the estimated marker positions, the calculation of the distances between the markers used to determine the labeling of each marker can be eliminated. It is because of this that the algorithm can be accelerated even further. This further acceleration is the result of the marker positions being estimated and tracked, having no chance of being confused with any other markers on the screen. Once the markers are labeled during the first three frames, the algorithm then tracks their labels along with the estimated and actual marker positions and never has to perform the labeling algorithm again. This advantage of tracking the labeled markers explains how the aircraft in question can now roll greater than 85 degrees and the marker still be labeled accurately. The reasoning behind Assumption #3 is now clear. If this assumption were to be violated in any way when the software is activated (within the first three frames), the left and right markers would be confused and would continue to be tracked in the confused manner. Again, the likelihood that the aircraft would be banked more than 85 degrees during this time is very low.

As previously mentioned, the range of X and Y coordinates used for creating the processing window is based upon a fixed square search area. This search area can be adjusted depending on the accuracy of the estimates being made. If the estimates are not well correlated with the actual positions, then the search area will need to be made larger to account for the lack of accuracy in the estimates. This search area size relies greatly on the processing speed of the MV computer system. If there is much lag between frames, then the motion information used to perform the velocity calculations may be inaccurate due to frequent motion changes between frames, when the computer is not 'looking'. If this is the case, the processing window will need to be made larger to accommodate for the appropriate conditions. Conversely, if the processing time is very fast, the motion of the markers can be very diverse in speed and direction and the processing window can be quite small while still finding the markers accurately. This is, of course, the desired condition.

This algorithm is the result of a build up of ideas leading to this solution. Many aspects of image processing have been used in this algorithm and these have already been mentioned earlier in Section 3.2 and Section 3.3. Instead of developing a micro level flowchart to detail the operation of this algorithm, a macro level flowchart, shown on the following page in Figure 4.13, will describe the process used to perform the Advanced K-Means Clustering and Tracking Algorithm. The Matlab code for the Advanced K-Means Clustering and Tracking Algorithm can be found in Appendix C.

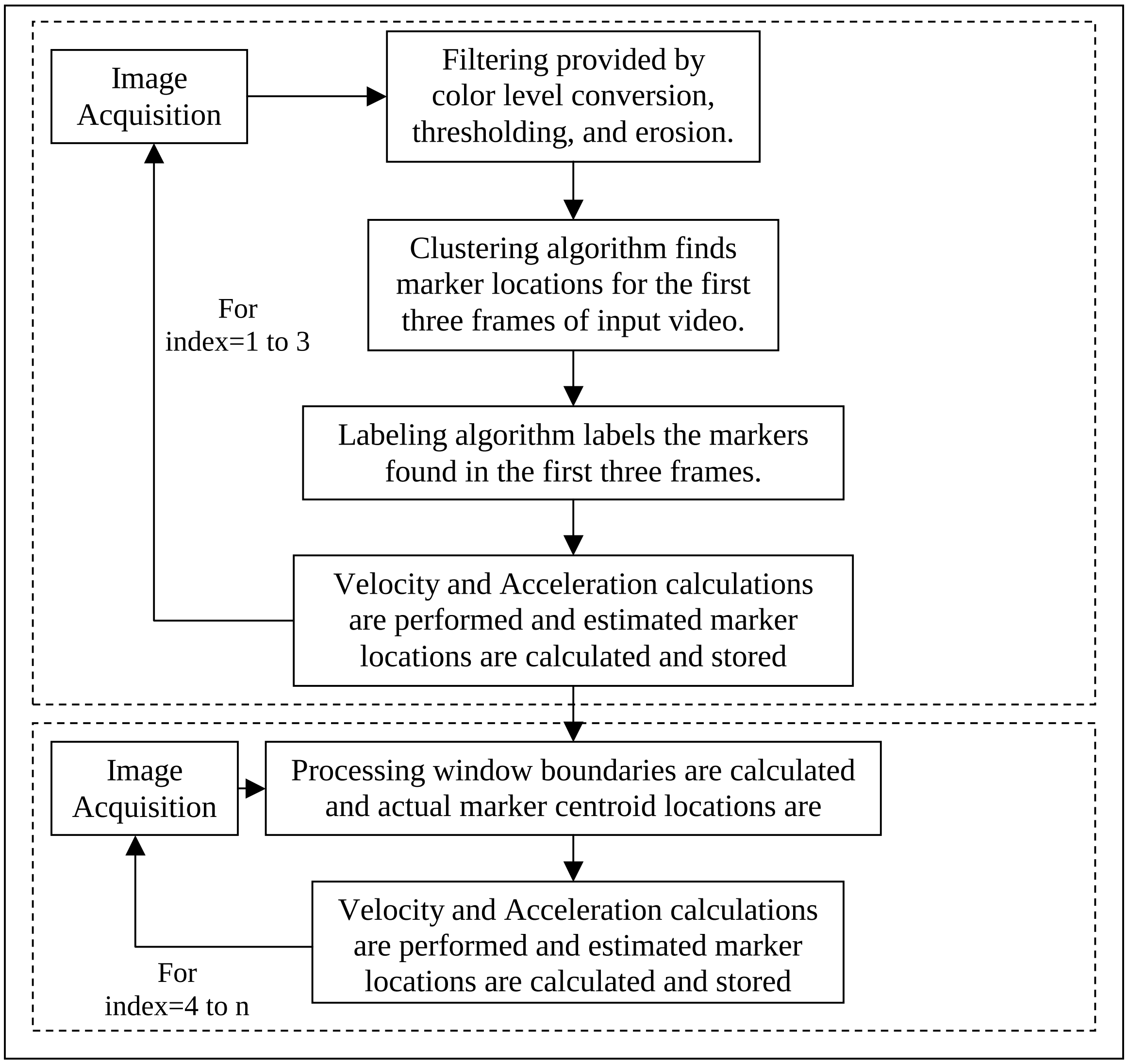


Image Acquisition
Filtering provided by color level conversion, thresholding, and erosion.
Clustering algorithm finds marker locations for the first three frames of input video.
For index=1 to 3
Labeling algorithm labels the markers found in the first three frames.
Velocity and Acceleration calculations are performed and estimated marker locations are calculated and stored
Image Acquisition
Processing window boundaries are calculated and actual marker centroid locations are
Velocity and Acceleration calculations are performed and estimated marker locations are calculated and stored
For index=4 to n

Figure 4.13: Macro Level Flowchart - Advanced Clustering and Tracking Algorithm

4.3 Experimental Procedures for the Runway Detection Problem

Like the Marker Detection and Tracking problem, a blend of hardware and software tools were used to address the goals associated with this problem. For this problem though, the software is not dependent upon the hardware setup at all. The only duty of the hardware is to provide an input video for the software to post-process. This made the construction of the software somewhat easier in that there were no internal hardware/software interaction issues to deal with. An overview of the experimental procedures required to address the Runway Detection problem is shown on the following page in Figure 4.14.

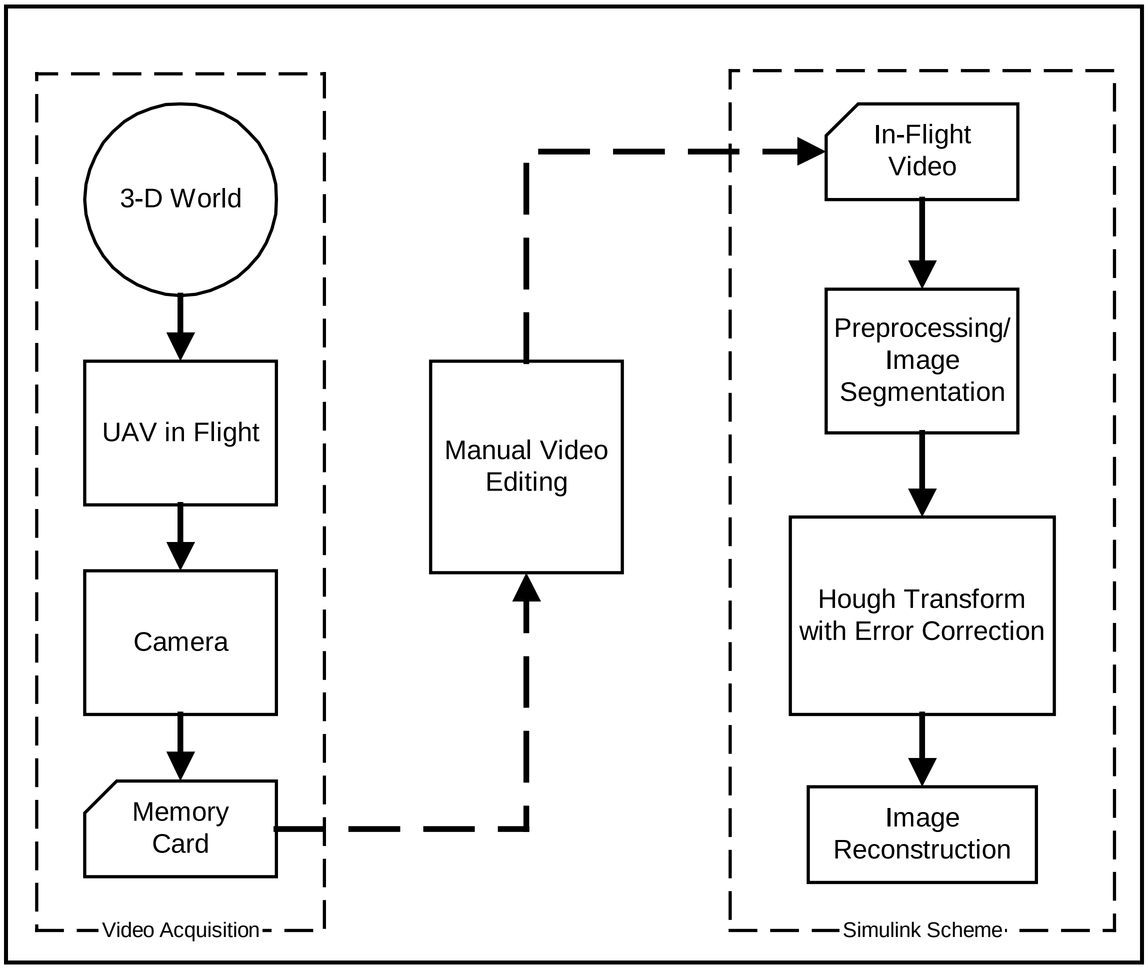


Figure 4.14: Runway Detection Experimental Procedures

4.3.1 Hardware Used for the Runway Detection Problem

This section is dedicated to the physical description and description of the function of each piece of equipment used to address the Runway Detection problem. Through this, the function, importance, and experimental procedures for each part as it relates to the problem will be outlined.

4.3.1.1 Description of Mustek DV-4000 Mini DV Camera

The Mustek DV-4000 Mini Digital Video (DV) Camera is the camera that was used on the aircraft while obtaining runway video to be used in a post-processing fashion in order to evaluate the runway detection scheme. This camera is perfect for this application because it is very lightweight, has an adequate field of view, and it can store a large amount of video enabling a long flight time. It is also quite small, which allowed it to be easily mounted on the aircraft test bed. Another thing that makes the camera a good candidate for this job is the fact that it is fairly low resolution which allows the software to be tested in a low resolution setting and it also enables the software to perform its best and fastest rate possible due to the small resolution of the video. A full description of the Mustek camera specifications is shown in Table 4.4.

Table 4.4: Mustek DV-4000 Mini DV Camera Specifications

Sensor Type	3 Mega pixel CMOS
Resolution	640 (W) by 480 (H)
Focal Length	8.5 mm
Focus Range	∞ - 0.2 m
Field of View @ 30 meters	12.71 m (W) x 9.64 m (H)
Iris Operation	Fixed @ F2.8
Focus Operation	Automatic
Frame Rate	10 fps
Video Format	MPEG-4
Capacity	> 3 hours recording time
Size	3.5” x 2.5” x 1.125”
Weight	118 g

This use of this camera created a real test for the software in terms of error correction. With its frame rate of only 10 fps and a highly dynamic field of view created by the nature of flight, the video taken by the camera is not the smoothest video ever encountered. This being the case, the difference in movement from frame to frame was sometimes great and this allowed the software to be put to the test using such a ‘jumpy’ and unstable video. Also, since this camera would automatically adjust the video gain, the brightness sometimes would vary due to the lens being pointed towards the sun, to the clouds, or ground. This also provided a great testing opportunity for the pre-processing/image segmentation part of the runway detection software to see how well it would perform with varying lighting conditions resulting in varying brightness. The Mustek DV-4000 Mini DV Camera is shown on the following page in Figure 4.15.

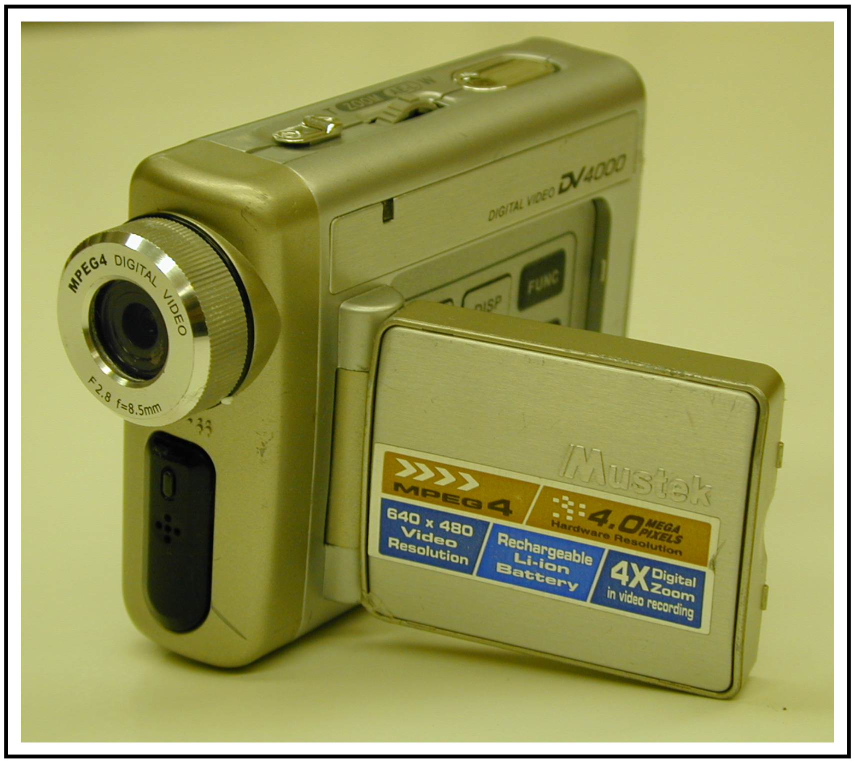


Figure 4.15: Mustek DV4000 Mini DV Camera

4.3.1.2 Cessna 152 Video Acquisition Platform

In order to achieve the first part of the experiment, a test bed must be selected to carry the video equipment in order to obtain the video of the runway. In this case, the Cessna 152 aircraft was selected as the test bed. The Cessna 152 is a large 35% scale replica of an actual Cessna 152. The payload capacity is enormous and as such, carrying a small camera is no huge task for it. The Cessna 152 specification are shown in Table 4.5, on the following page.

Table 4.5: Cessna 152 Specifications

Span	120”
Length	86”
Height	30”
Weight	34 lbs.
Payload Capacity	~10 lbs.
Duration	>30 minutes
Engine	Zenoah G-62 with Mejzlik 22x10 prop
Radio	JR XP9303 9 channel PCM radio system
Cruise Speed	~60 knots

The Cessna 152 test bed is shown below in Figure 4.16.

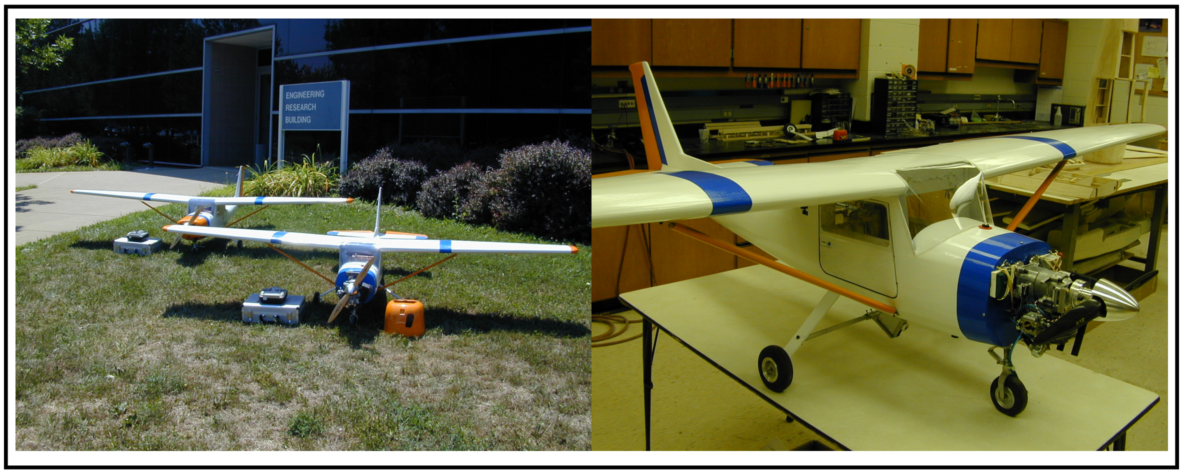


Figure 4.16: Cessna 152 Model Test Bed

The Mustek DV4000 Mini DV Camera was mounted on the Cessna 152 using a bracket that was custom designed and manufactured by the author. The bracket is a ½” solid aluminum rod which is cut and threaded on each end to match the angle of the original landing gear. The landing gear then had mounting holes (to match the threaded holes in the rod) drilled so that the rod could be mounted with socket head cap screws to the landing gear. The camera mount plate was engineered such that the camera could be rotated by loosening the bolts on the mounting collar and rotating the mount on the rod and then retightening the bolts. This would allow for different viewing angles to be achieved with one mount. In this configuration, the camera was set to recording mode before takeoff and was deactivated upon landing, thus capturing the entire flight.

Vibration was a concern, but turned out not to be an issue once the aircraft was in flight. The sturdiness of the bracket also helped this situation. The close up view of the DV4000 and the camera mount can be seen in Figure 4.17, below.

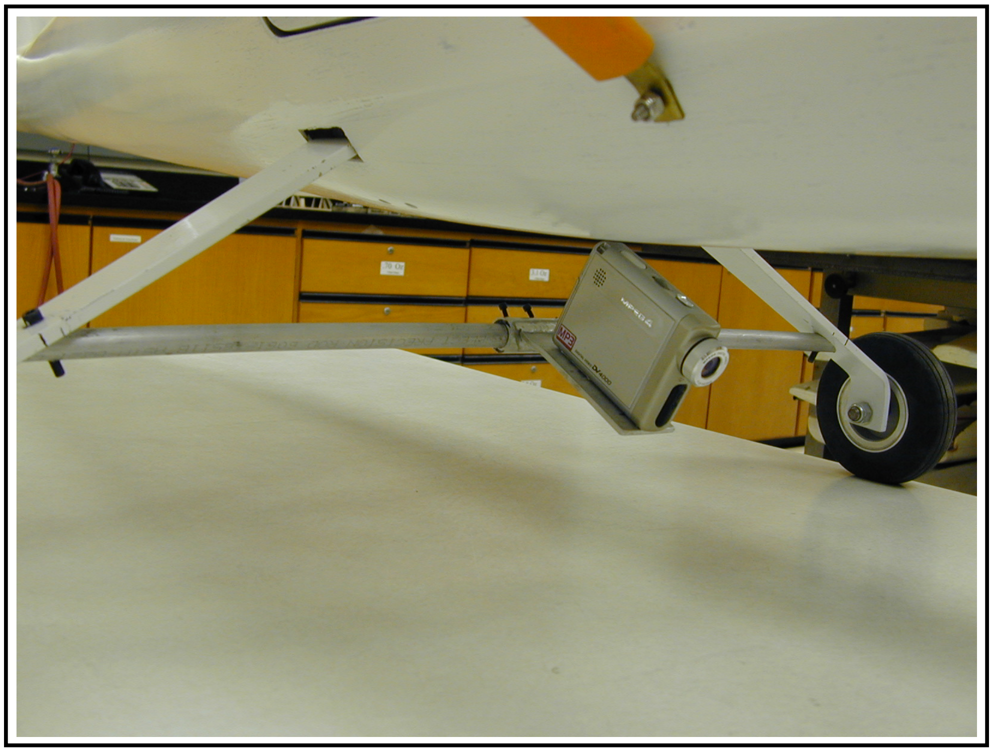


Figure 4.17: Close Up View of the DV4000 Camera Mounted on the Cessna 152

4.3.2 Software Used for the Runway Detection Problem

The software used to address the Runway Detection problem consists of only one method and that is the Simulink based method using the Hough transform. In order to apply the Hough transform to the problem of runway detection, many things have to come together. These include the acquisition and filtering scheme, the actual Hough transform operations, an error checking scheme to eliminate false peaks in the Hough space, and finally a scheme to put all of the images back together and display them. Each of these subsystems are clearly labeled and each will be fully explained in subsequent sections. Figure 4.18, on the following page, shows the main Simulink® scheme used to perform runway detection.

There are several main concepts to be discussed in this section. The image acquisition block is shown on the following page, in Figure 4.18, in magenta. This block will be discussed in Section 4.3.2.1. The image preparation, conversion, filtering and edge detection routines are contained in the pre-processing subsystem that is shown on the following page, in Figure 4.18, as the cyan colored block. The image acquisition and pre-processing subsystem will be discussed in Section 4.3.2.2. The Hough transform and its related operations are contained in the Hough transform operation subsystem which is shown on the following page, in Figure 4.18, as the light green colored block. The Hough transform operations subsystem will be discussed in Section 4.3.2.3. The Rho/Theta Correction block is a subsystem that contains a feedback routine capable of eliminating false peaks found in the Hough space. This block which is shown on the following page, in Figure 4.18, is colored in red. This subsystem will be explained in Section 4.3.2.4. The Image Regeneration block, on the following page, in Figure 4.18, colored in yellow, performs the task of putting all of the pieces of the original image back together so it can be displayed for visual evaluation. This subsystem will be discussed in Section 4.3.2.5.

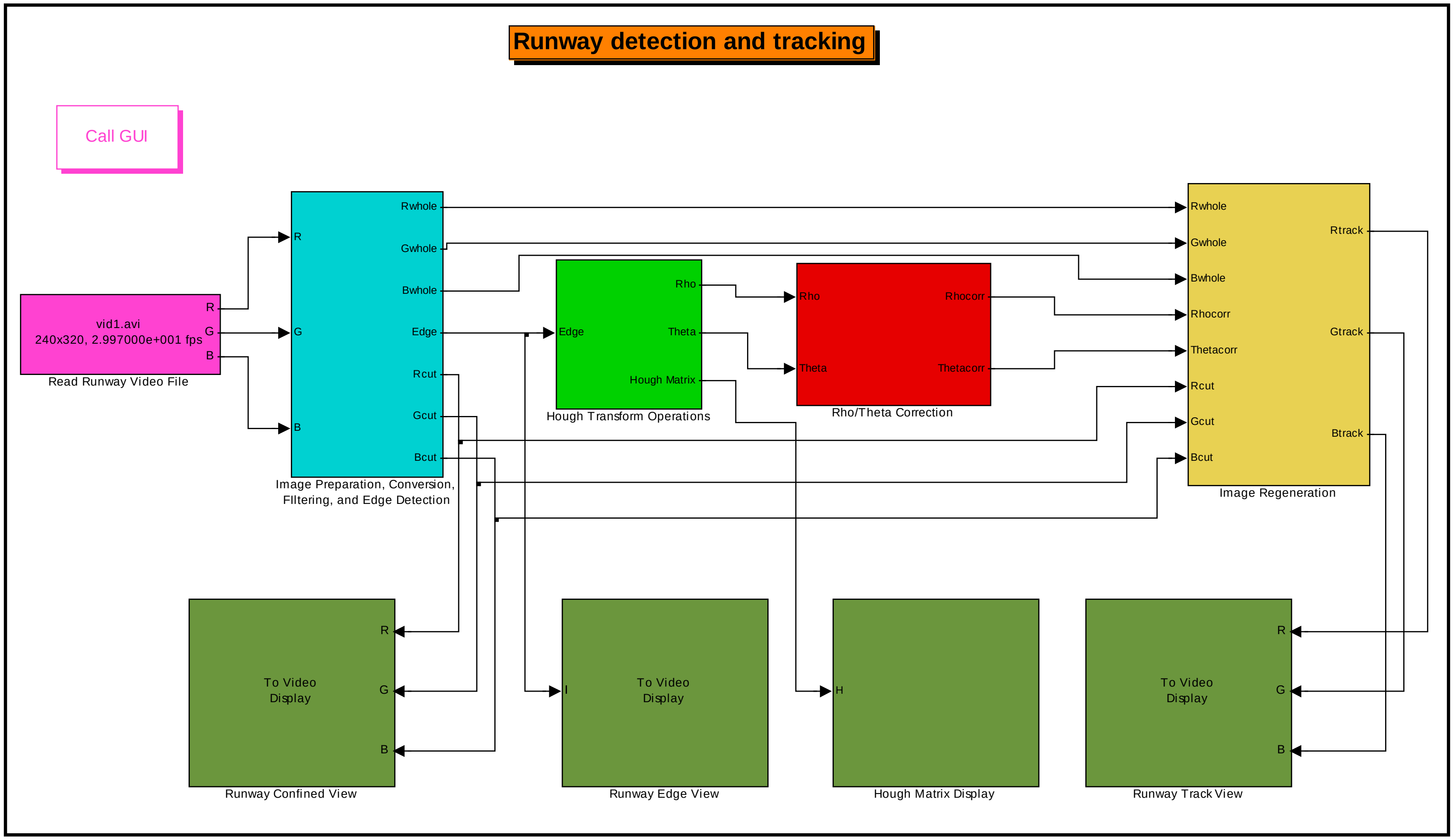


Figure 4.18: Runway Detection – Main Simulation System

4.3.2.1 Image Acquisition

The purpose of this scheme was to perform Runway Detection simulations on real camera images of a real runway. This simulation was restricted to the use of pre-recorded videos due to the lack of availability of an instrumented UAV to fly in order to record videos. As a result of this and other factors, the things that affect the flight and video characteristics have not been fully evaluated and the amount of usable video obtained is fairly small. Even though the video was small, it was sufficient to perform the simulations and to be able to evaluate the performance of the scheme. Therefore, the application of any other acquisition methods such as a simulated runway in a laboratory environment was not necessary and the small videos clips obtained from flight were used solely for evaluation of this scheme. A typical frame from a simulation input video is shown in Figure 4.19, below.

.

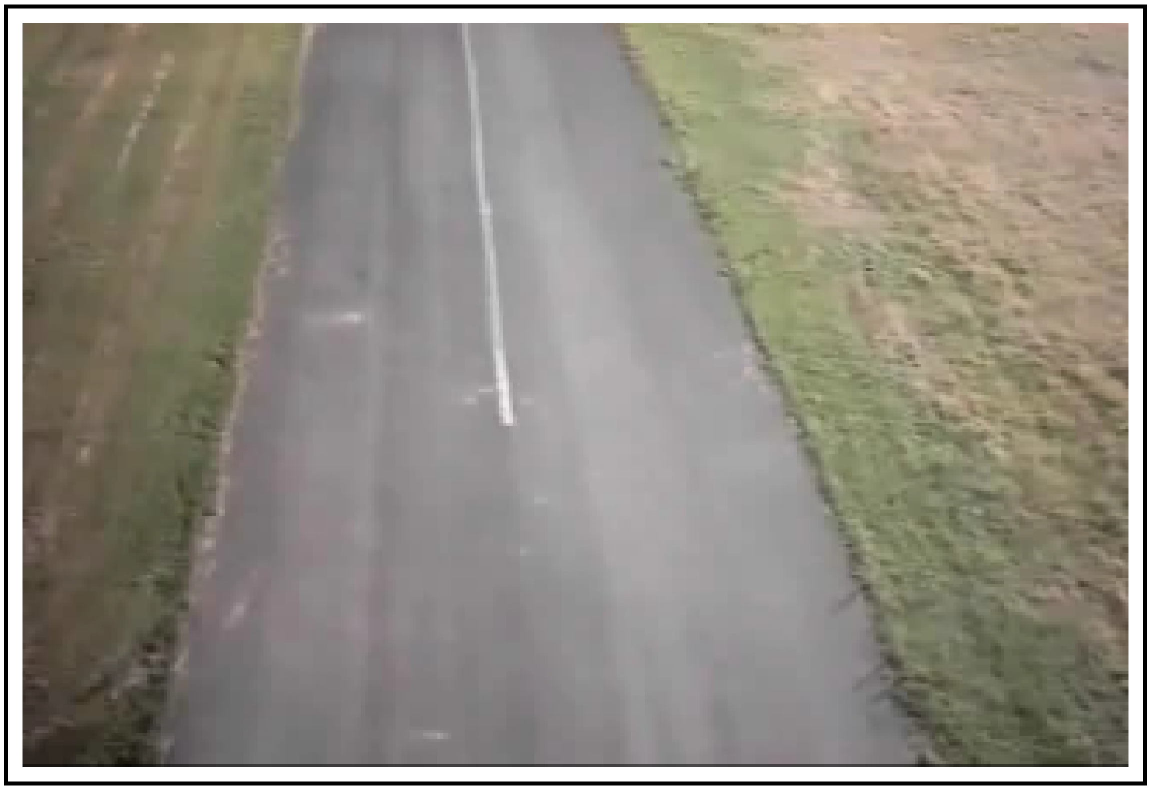


Figure 4.19: Runway Detection - Typical Input Image

4.3.2.2 Image Preparation, Conversion, Filtering, and Edge Detection

The image pre-processing subsystem, shown below in Figure 4.20, contains all of the functions necessary to convert the image from RGB to intensity,

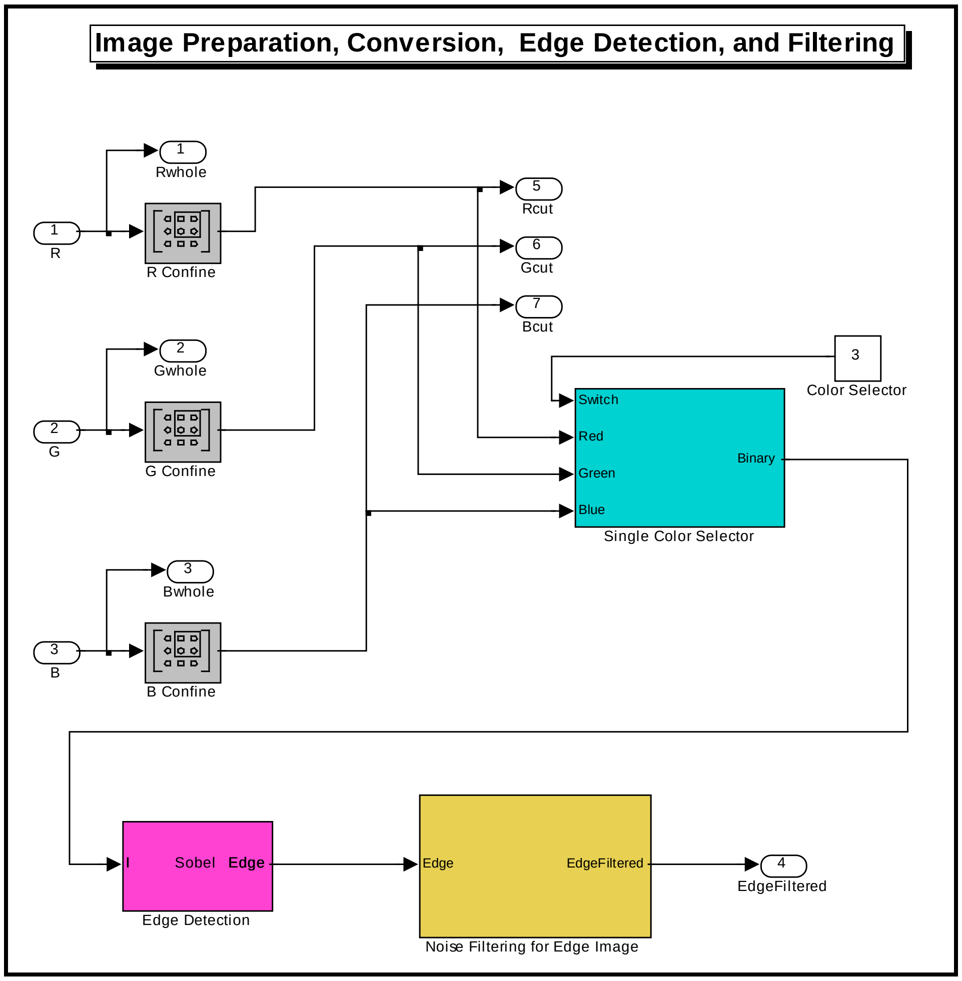


Figure 4.20: Runway Detection – Pre-Processing Subsystem

perform edge detection, and filter out most of the noise left in the image. The blocks used in this subsystem and all included subsystems are standard blocks within the standard Simulink® blockset[66] or the Video and Image Processing Blockset[60] within Simulink®. The inputs to this subsystem are the red, green, and blue components from the image acquisition block. The output is a fully filtered, binary edge image. This subsystem does contain one smaller subsystem. This subsystem is the Noise Filtering Routine, shown in Figure 4.20 as yellow block. There are some other very important functions that serve to speed up the processing time that will be discussed first. In Figure 4.20, on the left shown in light gray are three blocks that are labeled as ‘R Confine’, ‘G Confine’, and ‘B Confine’. These blocks are very essential to the efficiency of the scheme. These blocks take the full resolution image, which in this case is 320×240, and confines the image in the vertical direction, essentially picking a piece of the image out. The result is the same input image but it looks as though it has been cropped on the top and bottom. This allows approximately 40% of the image to be ignored while still being able to detect the lines on the runway. This could also be useful if the detection was involving something that was known to be in the same place in the image frame all of the time, such as the horizon. This would allow for almost 100% positive identification by ignoring all of the other lines in the image and only looking at a small area around it. The whole image is kept intact and sent out of the block for use later in the scheme as well as the ‘cut’ portion of the image. These will be needed later to put the image back together. It should be noted that the rows of the image to which the processing is confined is used definable in the GUI.

The image then enters the section that performs the color conversion. This is accomplished using the RGB to Intensity conversion which was discussed previously in Sections 3.2.3.2. Therefore, the details of the conversion will not be covered here. The RGB to intensity conversion is performed slightly different in this case. Instead of converting all of the colors to intensity, there is a color selector, indicated in Figure 4.20 as the cyan colored block. This color selector allows the user to switch between using only one of the colors at a time, depending on the conditions in the image. For example, if there were a lot of green in the image from a grassy field that was causing problems in the line detection, by simply turning green off and using red or blue, this problem can not be eliminated, but this is an action that helps remove the influence the green field is having on the resultant edge image. Therefore, by doing this, an actual RGB to Intensity color space conversion block is not necessary. Simply using one color is like having an 8-bit intensity image instead of having a 1-bit intensity image.

Once the color space conversion is complete and an intensity image has been obtained, the image is sent to the Sobel Edge Detection block shown previously in Figure 4.20, colored in magenta. The Sobel edge detection routine has previously been discussed in Section 3.4.1 and no further explanation will be given here. The edge image is then sent to the Noise Filtering subsystem block shown previously in Figure 4.20, colored in yellow. The Noise Filtering subsystem is shown below, in Figure 4.21. This subsystem is a simple morphological opening that was previously discussed in Section 3.4.2.4. The unique part of this filter lies in the structuring element. Since this scheme has the purpose of detecting lines that make up a runway and the lines in this setup run vertically through the image frame, a structuring element tailored to enhance vertical lines is used. The structuring element is a 3×3 matrix with the center column set to one and the rest of the element is zero. This greatly enhances the vertical lines on the runway and filters out the rest of the noise from the surroundings quite effectively.

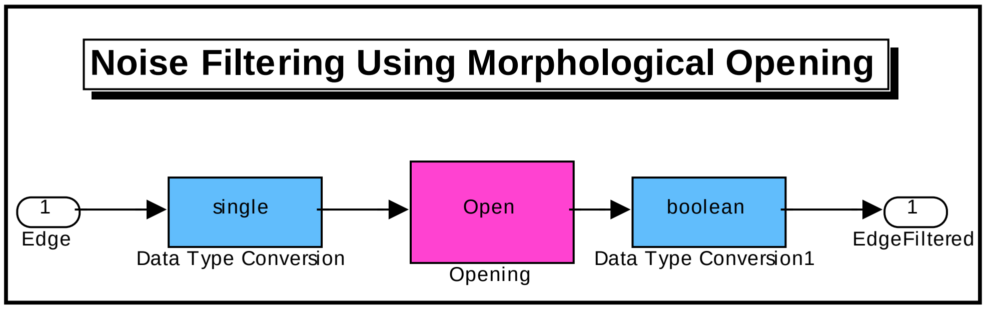


Figure 4.21: Runway Detection - Noise Filtering Subsystem

Figure 4.22, on the following page, shows a typical output image from the pre-processing subsystem.

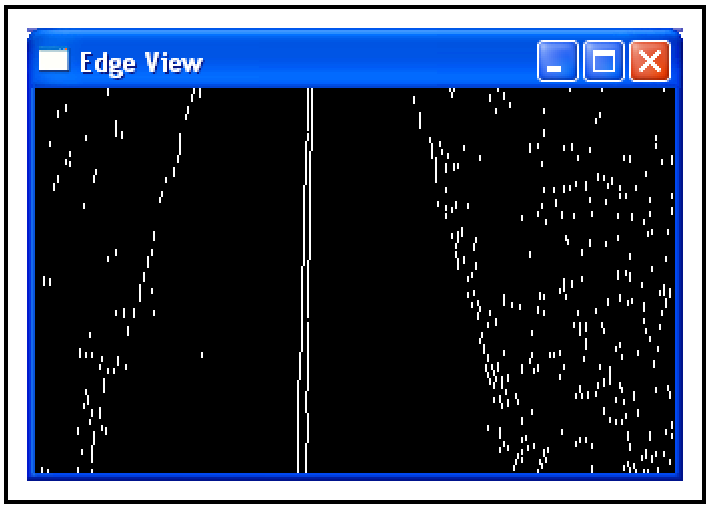


Figure 4.22: Runway Detection –Edge Image

4.3.2.3 Hough Transform Operations

The Hough transform subsystem, shown on the following page in Figure 4.23, contains only two higher-level blocks. These blocks are the Hough transform block and the Hough peaks block and both of these blocks were standard blocks included in the Video and Image Processing Blockset[60] within Simulink®. The Hough transform block as well as the Hough peaks block and their application has already been discussed in Section 3.4.3.1. These points will not be discussed again, but their inputs and outputs and the application within this particular scheme will be discussed.

The Hough transform operations subsystem uses the black and white edge image from the image pre-processing subsystem as its input. In particular, the Hough transform block within this subsystem takes this input directly. This block can be found in Figure 4.23 as the yellow colored block. The outputs of the Hough transform operations subsystem are the ρ values and the θ values that correspond to the number of Hough peaks desired. The Hough transform block has two options. These options are the rho resolution and the theta resolution to be applied while performing the transform. The ranges of these options are fixed within the block, therefore the only thing to determine is the resolution of these values and both of these values are user definable in the GUI.

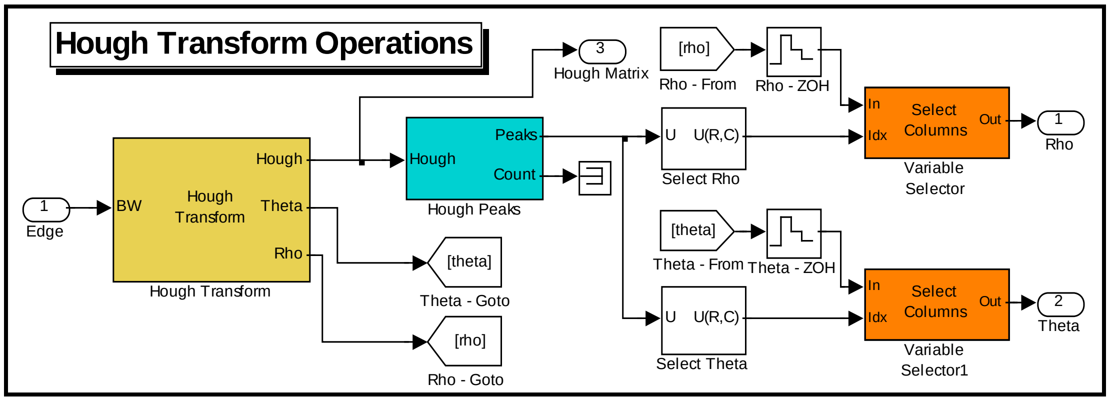


Figure 4.23: Runway Detection – Hough Transform Operations

The ρ and θ values are stored for later use and the Hough peaks block, which is found in Figure 4.23 as the cyan colored block, uses the Hough space or Hough matrix as its input. The Hough peaks block has three options that can be changed to alter its performance. These options are possibly the most influential values used in the entire scheme. The first option is the value of the desired number of peaks. The second option is the threshold value used in determining if a peak is actually a peak or not. The third option is the neighborhood size. The neighborhood size is the size of 'block' of the space in the Hough matrix that is searched to find a peak. In other words, once the block finds a peak, it checks the threshold value for the entire neighborhood size to ensure that it is a peak. The output of this block is the short list of the ρ and θ coordinates of the strongest peaks in the Hough space and the size of the list, of course, depends on the number of Hough peaks desired. The rest of the blocks shown in Figure 4.23 are blocks that help separate the ρ values from the θ values into their separate vectors from the output of the Hough peaks block. Figure 4.24, on the following page, illustrates the typical Hough space obtained when performing a runway detection simulation.

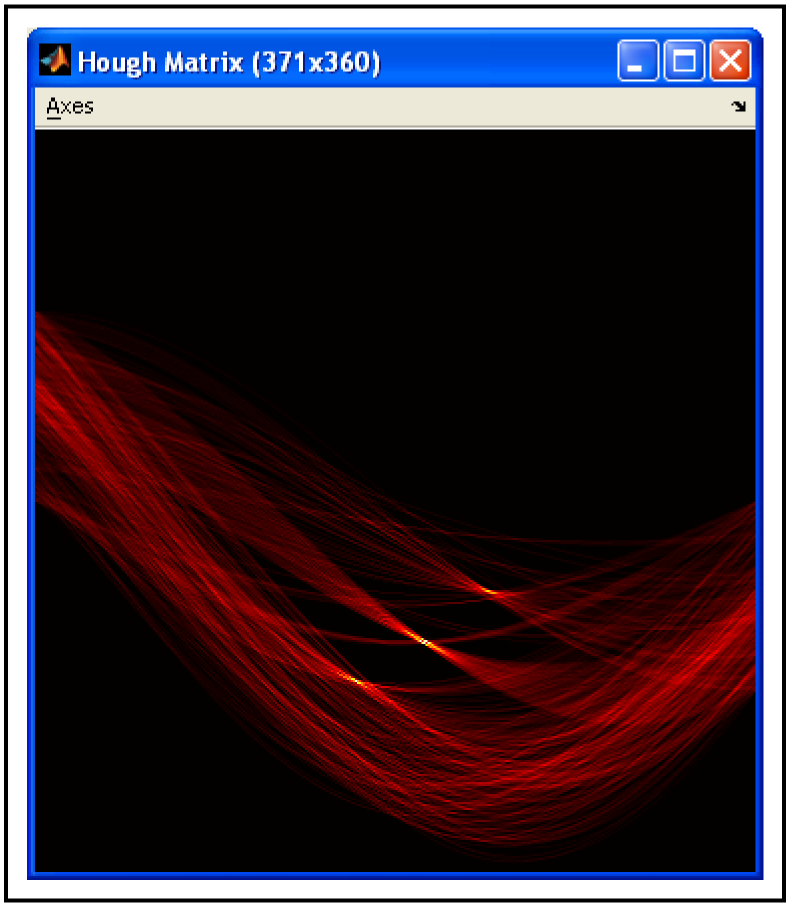


Figure 4.24: Runway Detection – Typical Hough Space

4.3.2.4 Rho/Theta Correction

The Rho/Theta Correction subsystem is shown previously in Figure 4.18 as the red colored block. It should be noted that all of the blocks in this subsystem are standard blocks found within the standard Simulink® blockset[66]. This block takes the ρ and θ values found from the Hough transform subsystem and basically checks to see if there is too much difference between the last values and the current values. If there is a great difference in the values from the last time step, the block assumes there has been an errant Hough peak used and it discards the current ρ and θ values and uses the values from the last time step. This is performed by using a negative feedback loop with a one time step delay and a threshold value for both ρ and θ. The decision then enters a 'For Iterator' subsystem that helps to select which ρ and θ value to output based on the error flag. These subsystems will be described in detail below. The Rho/Theta Correction subsystem is shown in Figure 4.25, on the following page.

In order to begin the detailed discussion of the Rho/Theta Correction subsystem, the underlying principle of the subsystem must be explained. In the first two time steps, no correction is being performed. This two-step buffer is meant to allow any transients to disappear, for the line detection to become established, and for a good set of ρ and θ values to become set. This two time step wait time was determined to be adequate by performing several simulations and watching the performance of the scheme. If less than two time steps were used then the scheme would have large errors at the start of the simulation and if more than two time steps were used, no notable change could be seen. Therefore, it was determined that more than two time steps are not needed and making the Rho/Theta Correction subsystem wait longer to become active has no benefit. Once this is complete, the correction process can begin. This principle is controlled by the two green blocks shown on the following page, in Figure 4.25. These blocks are called 'N-Sample Switch' and their job is to change their state after the specified number of time steps has passed. Once the desired number of time steps has passed, in this case two time steps, the switch will flip and the last known corrected values will be sent to the negative side of the summing junction in place of the current values.

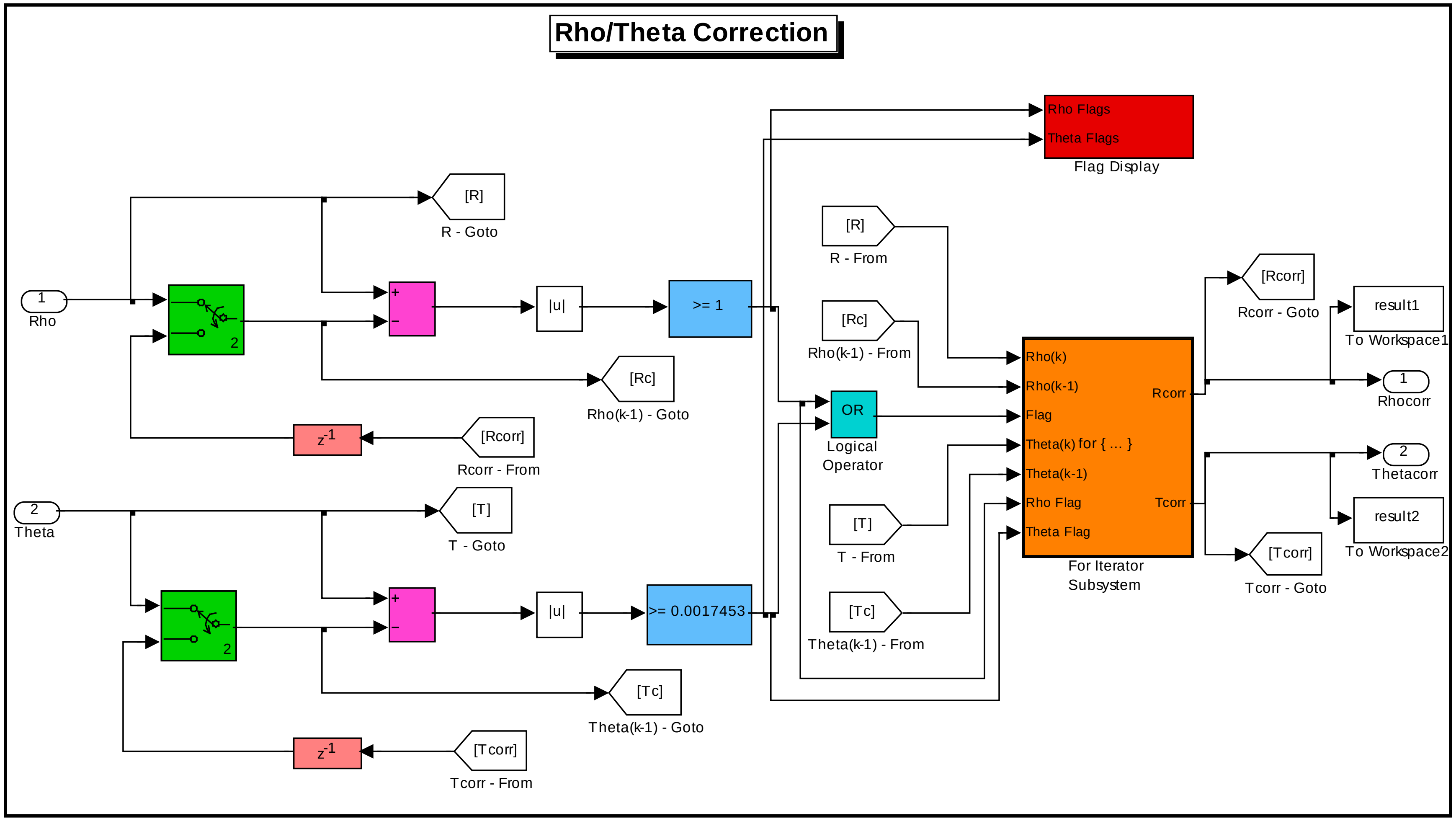


Figure 4.25: Runway Detection – Rho/Theta Correction Subsystem

Once this is complete, the values enter the light blue blocks found in Figure 4.25, which are called the ‘Compare to Constant’ blocks. These blocks are essentially comparing the error between ρ and θ values from the current time step and the last time step with an error threshold. If the error is greater than the threshold, then the block will output a one, if not a zero is the output. The output of these blocks are meant to allow the user to determine which line on the image is creating the most errors to give the user an idea of what to look at and what to adjust on the GUI to produce better performance. These flags are then fed into the ‘OR’ which is shown in cyan in Figure 4.25. The ‘OR’ block will output a true signal no matter which error flag is true. By using the ‘OR’ block, this ensures that almost under no circumstances will an errant set of values be passed on.

The situation at this point is that there is a set of flags indicating which Hough peaks may be false peaks. The problem is that there is more than one flag in a vector since there are more than one peak being used, in this case three peaks. Therefore, the ‘For Iterator’ subsystem is used. The ‘For Iterator’ subsystem block is shown in Figure 4.26, on the following page. This subsystem takes care of applying the decisions made by the ‘Compare to Constant’ blocks. The need for this subsystem is based on the need to make different decisions about each Hough peak individually in the same vector. Using this subsystem, the error flag vector is not looked at as a whole, but it is looked at as elements in the vector. The “For Iterator’ subsystem will loop itself through the decision making process for each of the error flags separately. This allows for one Hough peak to be acceptable and for another not to be acceptable in the same time step, thus applying the correction to only one of the Hough peaks and not the others. This is the heart of the Rho/Theta Correction subsystem and without it, the error corrections would have to be applied to all three Hough peaks or none at all. If this was the case, there would surely be no line to make it though the Rho/Theta Correction subsystem block without being corrected regardless if it was under the error threshold or not.

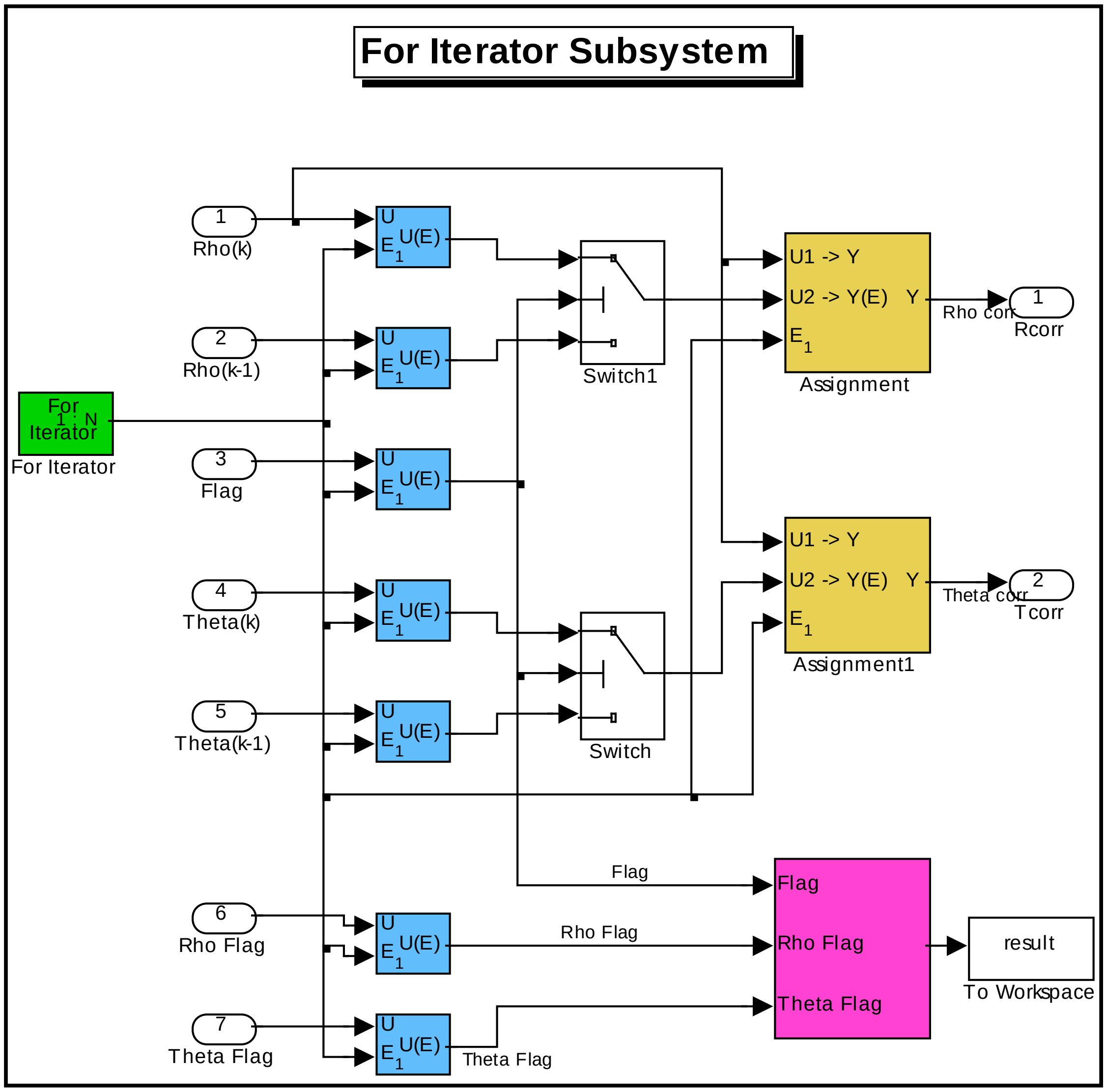


Figure 4.26: Runway Detection – For Iterator Subsystem

Even though the error flag is based on the 'OR' of the comparison results, meaning one value could be acceptable and the other not acceptable, both the ρ and θ values are corrected. This can be seen in the setup of the switches in Figure 4.26, above, that control which value exits the block, the value from the last time step or the current time step. The error flag operates both switches at the same time, applying the correction to both values. The values are then assigned back into their original positions in the ρ and θ vectors so as not to be confused and they are sent to the output, which is in turn the output of the Rho/Theta Correction subsystem. The flag values are also stored for every simulation so as to aid in tuning the error thresholds and to determine if the simulation is really performing the best it can. Although the number of times the flags are true is not a perfect indication as to the performance of the simulation, it is a point of concern and it is counted and displayed in the results section of the GUI to be discussed later.

There is one more subsystem included in the Rho/Theta Correction subsystem and that is shown in red in Figure 4.25. This block is called the Rho/Theta Flag Display and it performs no essential duties in the scheme. It is an important analysis tool though, which allows the user to examine the video outputs and inputs while at the same time viewing which error flags are being set to true and false in the real simulation timeframe. This helps in determining which of the many settings in this scheme need to be adjusted in order to cause the number of times the correction is applied to be reduced. The optimal case in this simulation is when the output lines on the video track well with the actual lines visually and that the lowest number of correction flags are seen at the same time. The Rho/Theta Flag Display subsystem is shown in Figure 4.27, below.

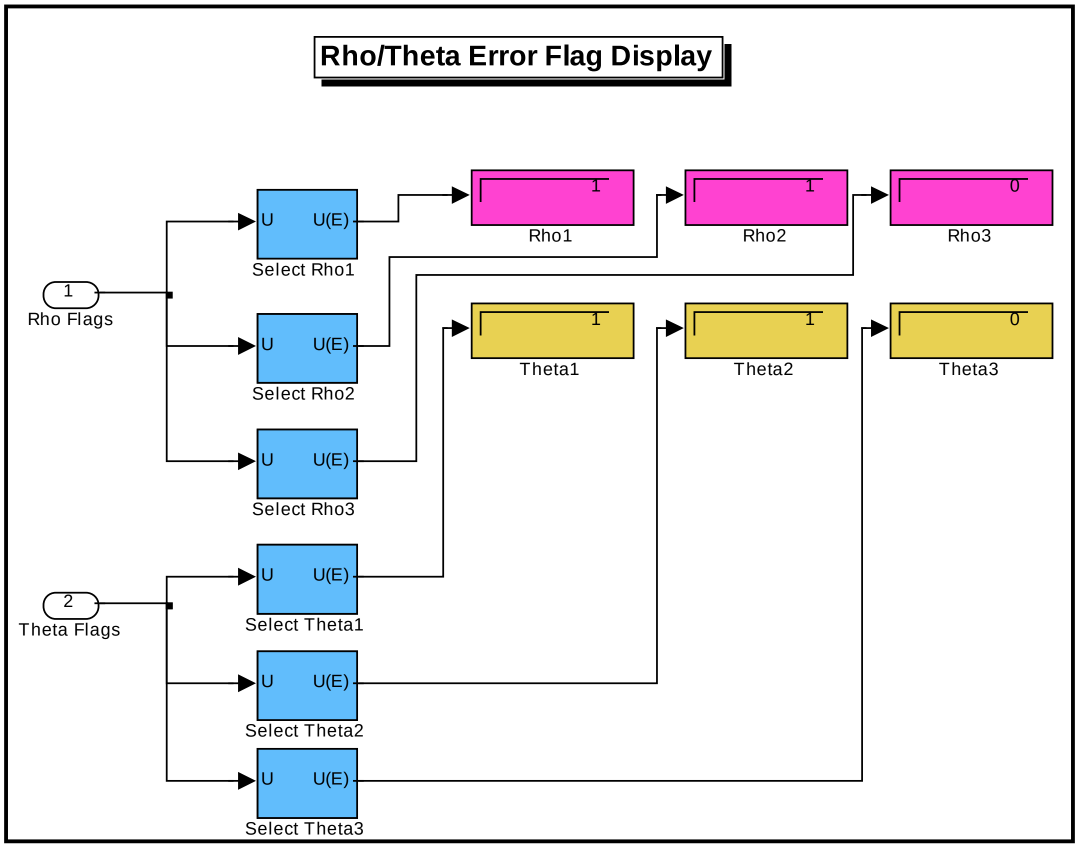


Figure 4.27: Runway Detection – Rho/Theta Error Flag Display

4.3.2.5 Image Regeneration

The Image Regeneration subsystem block plays an important role in this scheme helping the user determine if the software is performing adequately. The blocks used in this subsystem and all included subsystems are standard blocks within the standard Simulink® blockset[66] or the Video and Image Processing Blockset[60] within Simulink®. Recall from Section 4.3.2.2 when the image was confined to a smaller part, a horizontal slice if you will, which is the only part of the image that is processed. Therefore, the output image from the rest of the scheme is only this slice. This block takes care of joining the parts of the image (top, middle slice, and bottom) back together so the user can see the image in its entirety while overlaying the Hough lines in Cartesian space on the image. The Image Regeneration block is shown previously in Figure 4.18, as the yellow colored block. There are many inputs to this block as can be seen in this figure. The original image and the cut portion of the image are both needed to regenerate the original image and the ρ and θ vectors are needed to calculate the Cartesian coordinates of the lines corresponding to the Hough Peaks that were found. The output is simply the red, green, and blue images that make up the regenerated image. These outputs are connected to a video display for visual reference. The Image Regeneration subsystem is shown on the following page, in Figure 4.28.

In this subsystem, there are four separate processes happening. First, the red, green, and blue parts of the original whole image are confined in such a way to cut the middle part that was used for line detection away from the image, leaving only the top and bottom pieces. This can be seen in Figure 4.28 by looking for the confine blocks colored in gray. The second thing is that the Hough Lines block, colored in green in Figure 4.28, is using the middle cut piece of the image from the pre-processing subsystem and the ρ and θ vectors to perform an inverse Hough transform and thus calculate and plot the resulting detected lines on that cut image piece. Third, the top, bottom, and middle piece with the detected lines overlaid on it are rejoined by using the vertical concatenation blocks shown in Figure 4.28, colored in yellow. The result from the vertical concatenation blocks is the final product of the scheme.

The fourth thing being performed in this subsystem is the execution of yet another subsystem called the Hough Lines Calculator subsystem. This subsystem is responsible

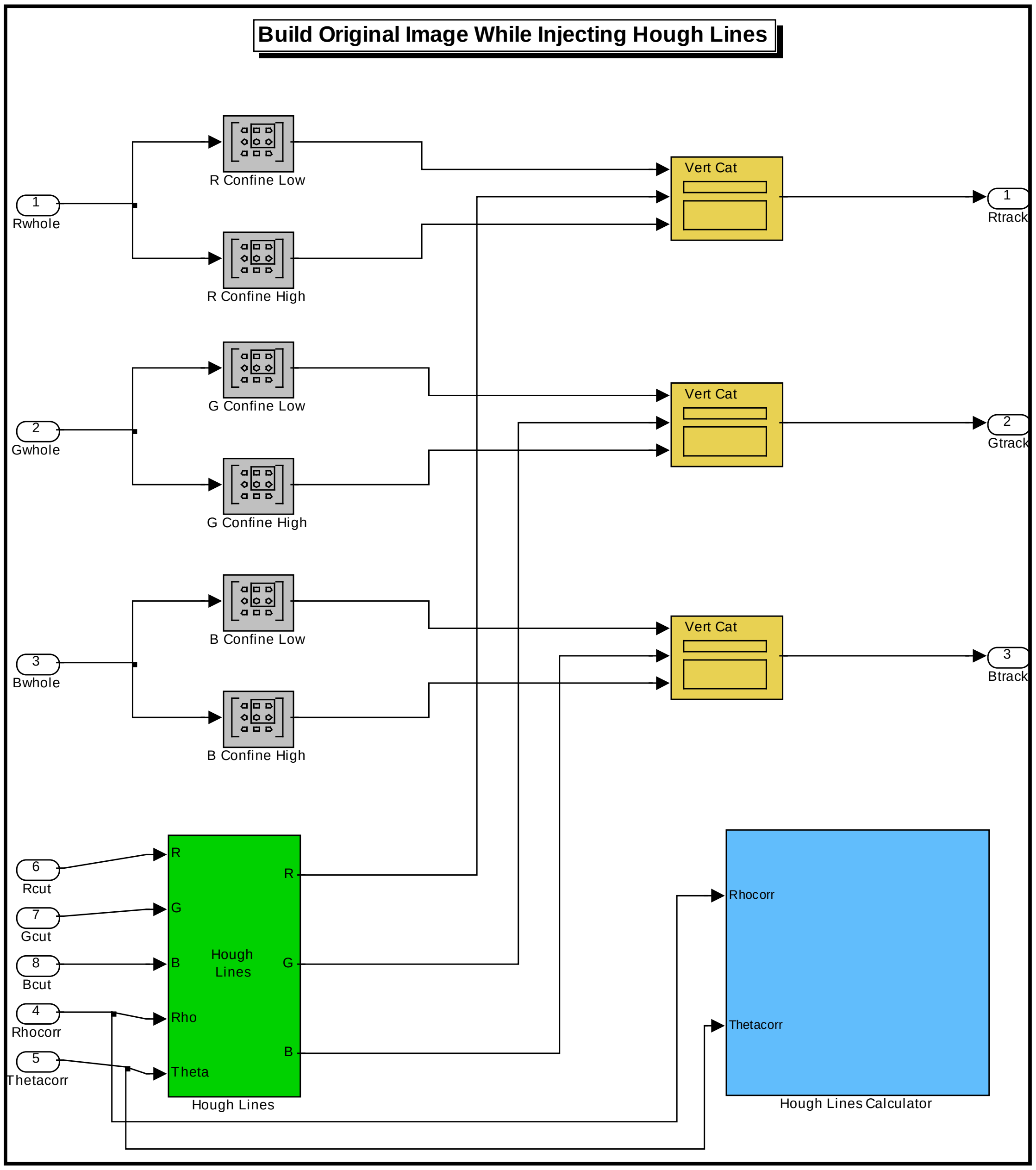


Figure 4.28: Runway Detection – Image Regeneration Subsystem

for verifying that the Hough Lines block is actually performing its assigned job. This is implemented as a second verification to the fact that the scheme did find the desired lines and that it is plotting them in the correct place. This subsystem essentially performs the same job the Hough Lines block does but its outputs, which are the end points of the lines in Cartesian space, are stored and later plotted in a Matlab plotting window for verification of the position of the detected lines. This block has no higher-level functions in it as it only performs a mathematical calculation. The Hough Lines Calculator subsystem is shown in Figure 4.29, below.

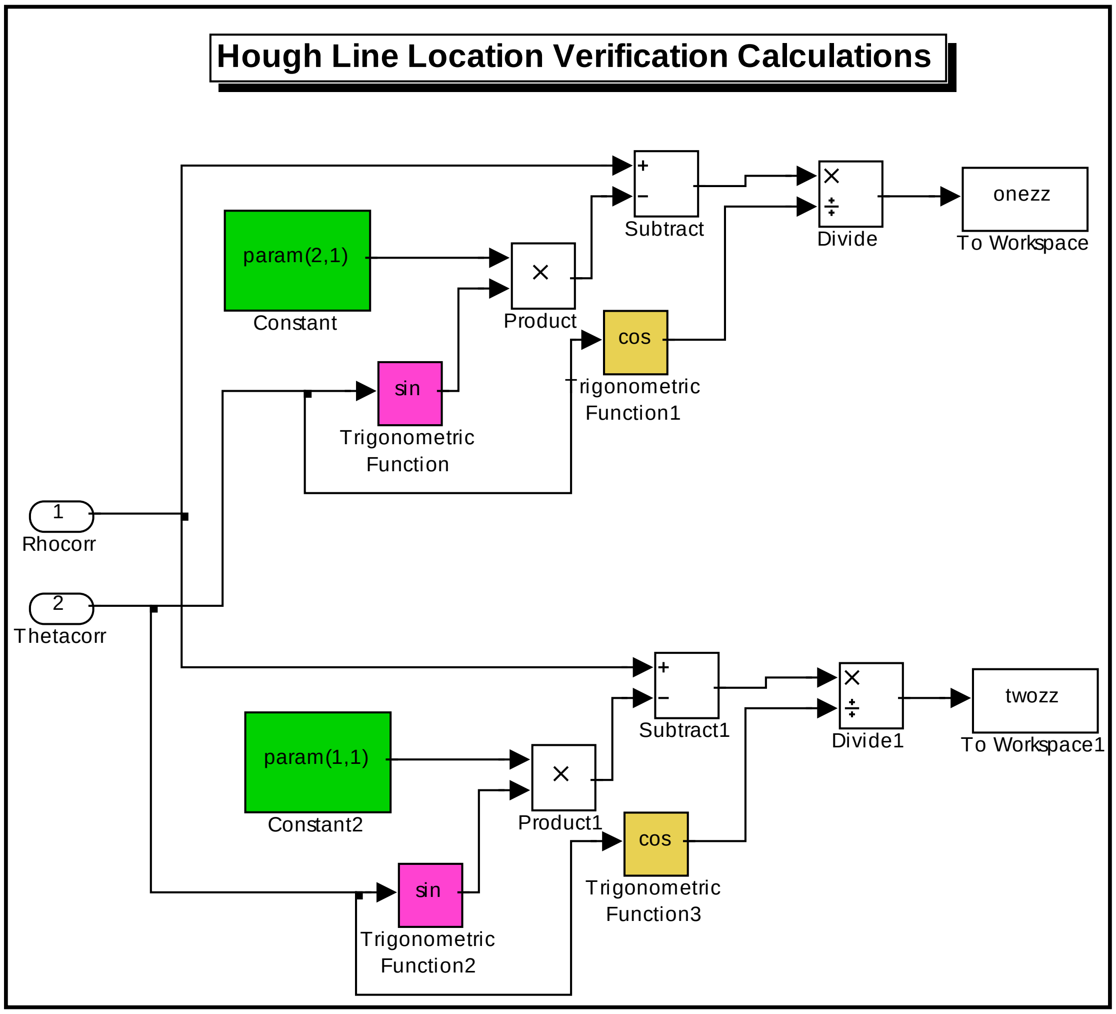
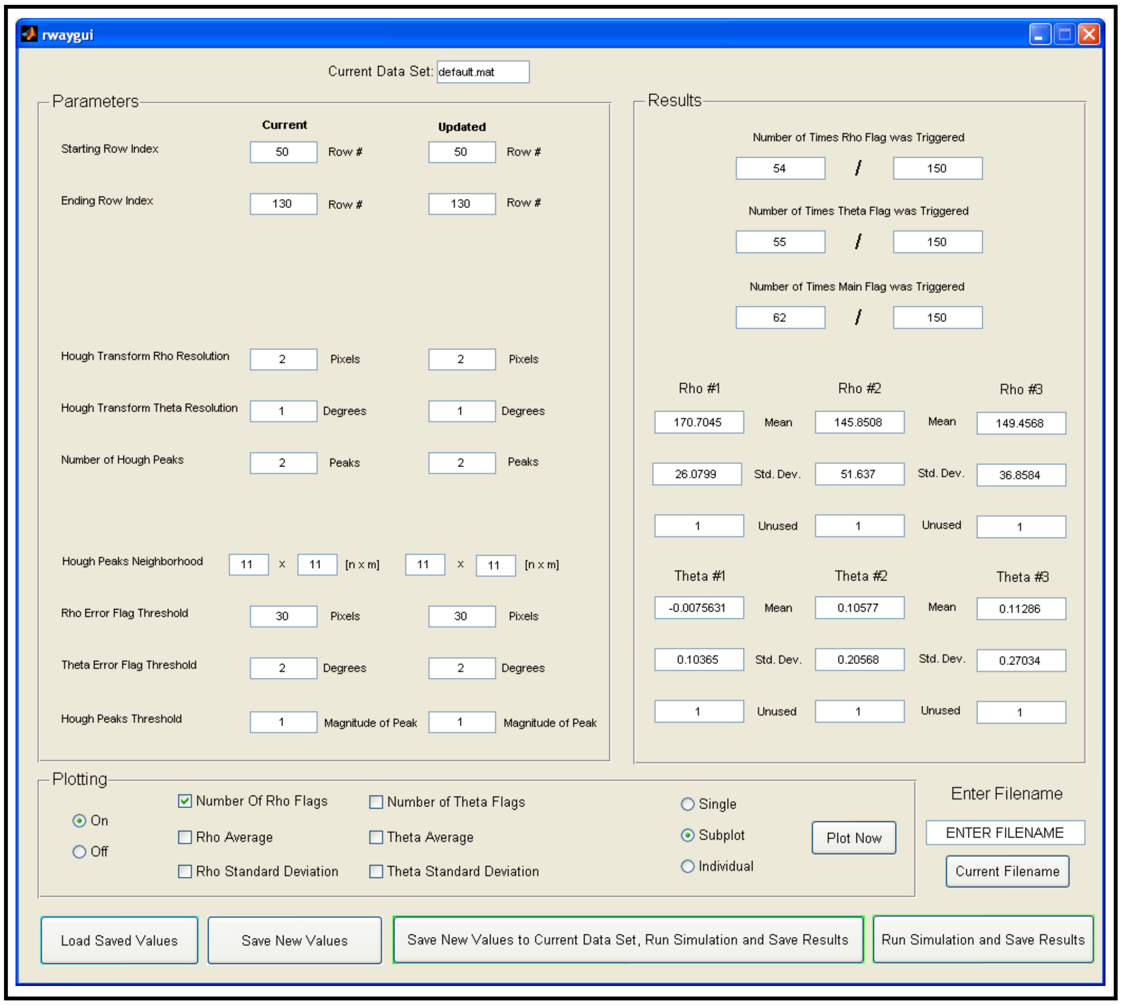


Figure 4.29: Runway Detection – Hough Lines Calculator Subsystem

4.3.3 Description of the Graphical User Interface

This section will explain the features and functions available in the graphical user interface (GUI) which was created for use with the Runway Detection scheme. The GUI allows many options useful for keeping track of simulation results and different sets of inputs as well as giving the user an easy to use plotting interface to view the results. Now that the entire scheme has been explained, the simulation inputs will be covered in Section 4.3.3.1. The result values display section is the section of the GUI which displays the counts on the error flags and this will be covered in Section 4.3.3.2. The trend plotting section is the section of the GUI which allows any number of used selectable plots to be made after a simulation is complete and it will be covered in Section 4.3.3.3. The final section of the GUI to be covered is the video analysis windows. These video windows show the simulation video at several stages throughout the process so the user can adjust parameters to fine tune the performance or simply view the output. The video analysis windows will be covered in Section 4.3.3.4. The entire GUI is shown below in Figure 4.30.



Current Data Set: default.mat

Parameters

	Current		Updated	
Starting Row Index	50	Row #	50	Row #
Ending Row Index	130	Row #	130	Row #
Hough Transform Rho Resolution	2	Pixels	2	Pixels
Hough Transform Theta Resolution	1	Degrees	1	Degrees
Number of Hough Peaks	2	Peaks	2	Peaks
Hough Peaks Neighborhood	11 x 11	[n x m]	11 x 11	[n x m]
Rho Error Flag Threshold	30	Pixels	30	Pixels
Theta Error Flag Threshold	2	Degrees	2	Degrees
Hough Peaks Threshold	1	Magnitude of Peak	1	Magnitude of Peak

Results

Number of Times Rho Flag was Triggered: 54 / 150

Number of Times Theta Flag was Triggered: 55 / 150

Number of Times Main Flag was Triggered: 62 / 150

Rho #1		Rho #2		Rho #3
170.7045	Mean	145.8508	Mean	149.4568
26.0799	Std. Dev.	51.637	Std. Dev.	36.8584
1	Unused	1	Unused	1

Theta #1		Theta #2		Theta #3
-0.0075631	Mean	0.10577	Mean	0.11286
0.10365	Std. Dev.	0.20568	Std. Dev.	0.27034
1	Unused	1	Unused	1

Plotting

[✓ On] Off

[✓ Number Of Rho Flags] Number of Theta Flags

Rho Average Theta Average

Rho Standard Deviation Theta Standard Deviation

Single [✓ Subplot] Individual

Plot Now

Enter Filename: ENTER FILENAME

Current Filename

Load Saved Values | Save New Values | Save New Values to Current Data Set, Run Simulation and Save Results | Run Simulation and Save Results

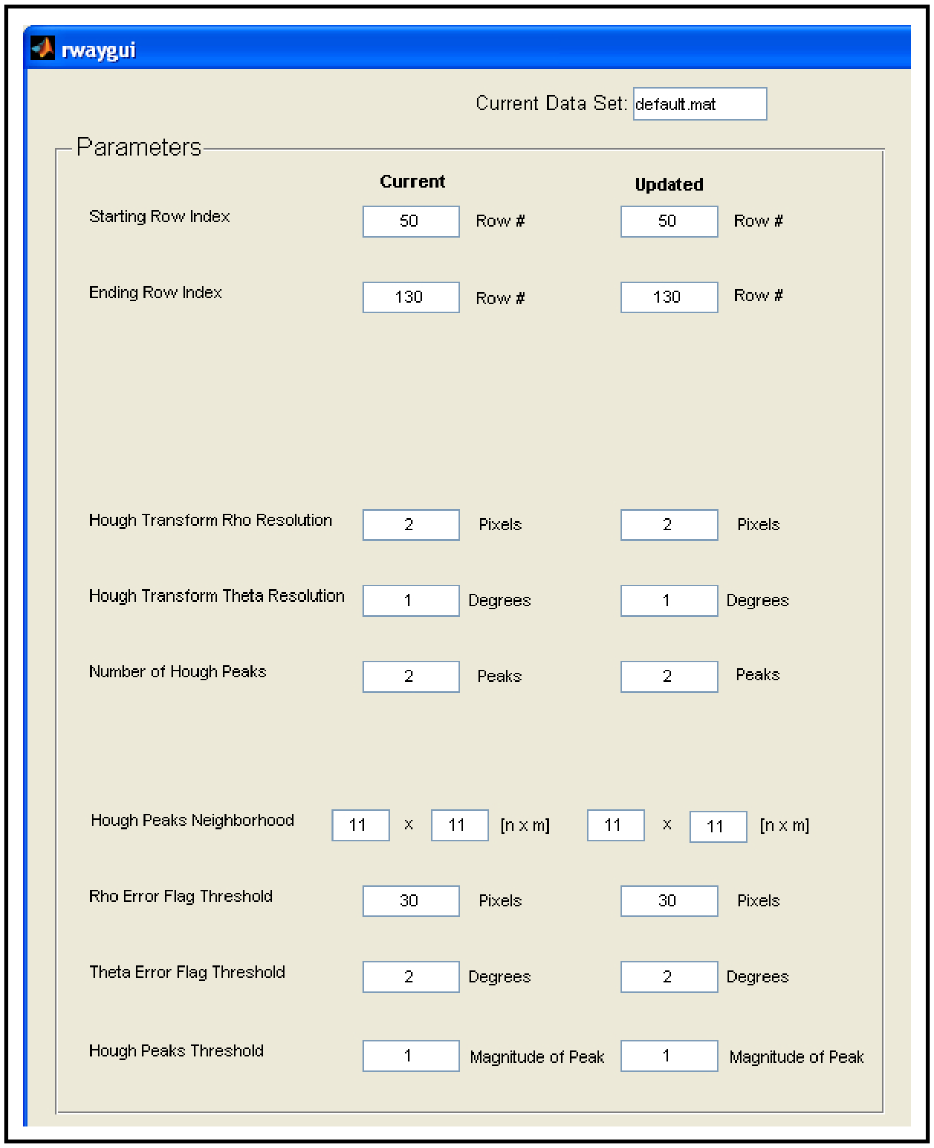
Figure 4.30: Runway Detection – Graphical User Interface

4.3.3.1 Simulation Inputs

The input section of the GUI is very straight forward. First, when the GUI loads up, it automatically initializes all of the parameters in the software with a default set of values. The GUI also gives the user the ability to load and save sets of parameters so that a simulation can be run and then the exact same setup can be recalled and performed again without the user having to know anything but a filename. The GUI parameters section consists of two columns, the first of which indicate the current value associated with the current data set loaded and the second is an editable box which allows the values to be changed. Once any value is changed, it is automatically updated in the workspace without the need to save the setup. This allows things to be changed quickly and the simulation ran for a trial and then if the outcome is acceptable the user may then want to save the setup. This keeps the user from saving a lot of junk setups in the phase of testing when major tuning of the parameters is taking place. The GUI also allows the user to run a simulation and not save the results or to run and save the results. When results are saved, the GUI also saves all of the information regarding the setup as well as the saved setup filename if there is one. The parameters section of the GUI is shown on the following page in Figure 4.31 and a description of each of the simulation inputs is provided below.

The following list is a description of each simulation input parameter:

1. *Starting Row Index* – The row in the image where the upper image confinement takes place. The image confinement block in Figure 4.20 requires this input to set the row of the image where the upper image confinement occurs.
2. *Ending Row Index* – The row in the image where the lower image confinement takes place. The image confinement block in Figure 4.20 requires this input to set the row of the image where the lower image confinement occurs.
3. *Hough Transform Rho Resolution* – Resolution used for the discretization of the ρ vector used in the Hough Transform block in Figure 4.23. This number determines the interval between ρ values used in the Hough transform in Equation 3.16.


rwaygui

Current Data Set: default.mat

Parameters

	Current		Updated	
Starting Row Index	50	Row #	50	Row #
Ending Row Index	130	Row #	130	Row #
Hough Transform Rho Resolution	2	Pixels	2	Pixels
Hough Transform Theta Resolution	1	Degrees	1	Degrees
Number of Hough Peaks	2	Peaks	2	Peaks
Hough Peaks Neighborhood	11 x 11	[n x m]	11 x 11	[n x m]
Rho Error Flag Threshold	30	Pixels	30	Pixels
Theta Error Flag Threshold	2	Degrees	2	Degrees
Hough Peaks Threshold	1	Magnitude of Peak	1	Magnitude of Peak

Figure 4.31: Runway Detection – Graphical User Interface Input Parameters

4. *Hough Transform Theta Resolution* – Resolution used for the discretization of the θ used in the Hough Transform block in Figure 4.23. This number determines the interval between θ values used in the Hough transform in Equation 3.16.
5. *Number of Hough Peaks* – This represents the number of Hough peaks that to be found in the Hough space. This value is used in the Hough Peaks block in Figure 4.23. Essentially, this represents the number of lines of interest in the input image.

6. *Hough Peaks Neighborhood* – The size of the 'window' of pixels that the Hough Peaks block in Figure 4.23 searches for when finding a peak. If this value is too large, the existence of two equally sized peaks within the window may occur and cause them to be ignored as individual peaks. These values should be smaller than the average distance between peaks in the Hough space when the scheme is applied to a specific application to ensure that the peaks will be detected reliably. These values must also be odd numbers for the searching of the neighborhood to work properly.
7. *Rho Error Flag Threshold* – The threshold used in the Rho/Theta Correction subsystem in Figure 4.25 for determining if there is excessive error in the ρ signal. This threshold must be tuned be examining the number of times the error flag is tripped and the visual performance of the algorithm. If the error threshold is too small, more correction than are necessary can take place and cause larger errors in the algorithm than is being corrected. This situation can be detected by visual examination of the output of the scheme for lines that do not change position on the display with respect to the actual lines in the image. This means that the same line is being fed back over and over because the threshold is too low. If the threshold is too high, not enough error correction will occur and when there is actually an error to be corrected, the scheme will overlook it and continue without feeding back any corrected lines. This situation can also be detected by visual examination of the output of the scheme for lines than change position by a great amount on the screen to a position that does not coincide with the desired output.
8. *Theta Error Flag Threshold* – The threshold used in the Rho/Theta Correction subsystem in Figure 4.25 for determining if there is excessive error in the θ signal. This threshold must be tuned be examining the number of times the error flag is tripped and the visual performance of the algorithm. If the error threshold is too small, more correction than are necessary can take place and cause larger errors in the algorithm than is being corrected. This situation can be detected by visual examination of

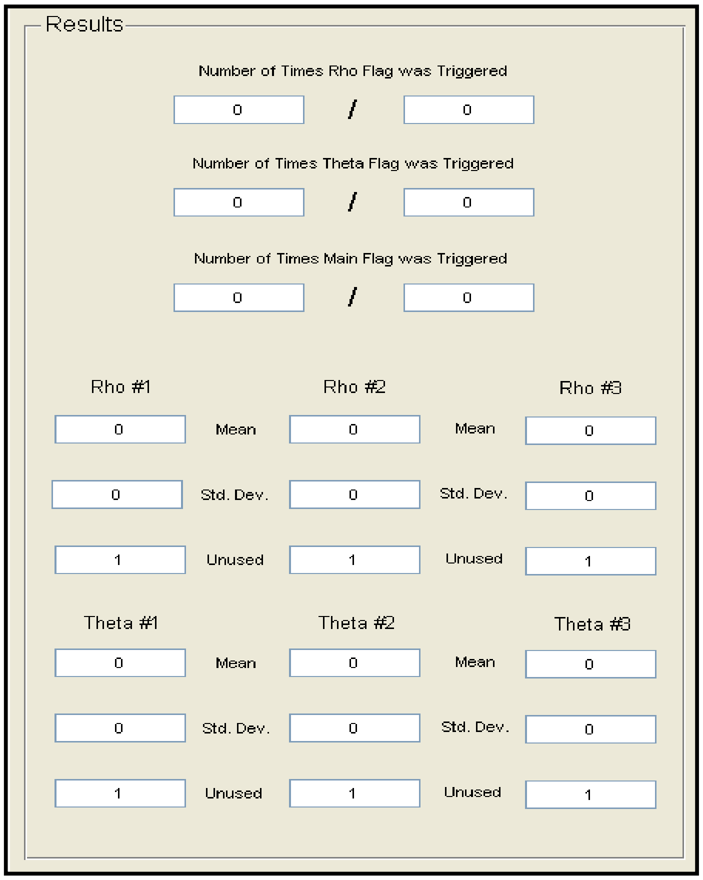
the output of the scheme for lines that do not change position on the display with respect to the actual lines in the image. This means that the same line is being fed back over and over because the threshold is too low. If the threshold is too high, not enough error correction will occur and when there is actually an error to be corrected, the scheme will overlook it and continue without feeding back any corrected lines. This situation can also be detected by visual examination of the output of the scheme for lines than change position by a great amount on the screen to a position that does not coincide with the desired output.

9. *Hough Peaks Threshold* – The threshold used by the Hough Peaks block in Figure 4.23 for determining if a peak is, in fact, a peak by examining its ‘height’. If height is above this threshold with respect to the pixels in the Hough Peaks Neighborhood, then it is a peak. Essentially, this value sets the minimum strength of a peak that is necessary to trigger this block to output that specific location as the location of a peak in the Hough space. If a potential peak is lower than this threshold, it will not be defined as a peak. The setting of this value is contingent upon the strength of the line definition in the input image. If the line is very clearly defined, the peak will be very strong and a high number (>15) may be used to ensure that the desired peak is being detected. If the line is not very clearly defined, the peak will not be strong, instead, it will look more like a hill than a peak and some tuning of both the Hough Peaks Neighborhood and this Hough Peaks Threshold should be done to ensure robustness with respect to finding the peaks in a situation when the lines are not so clearly defined.

4.3.3.2 Result Values Display Section

The result values display section is a part of the GUI which displays some stored and some calculated values which reflect on the performance of the runway detection scheme. At the top of this section, the number of flags for the ρ, θ, and the combined flag value taken after the OR block in the rho/theta correction subsystem. These are fairly good indicators of the performance for the scheme. It would be a perfect situation if the number of flags read zero, which would mean the scheme perfectly tracked the lines on every frame. But, this is never the case. The typical values for this will be presented in the Results section.

The other half of the result values display section is a section which displays calculated values which indicate the average and standard deviation of the ρ and θ values for each of the three lines in the image. This is not as good of an indication to performance as is the flag data, but it does give some indication as to the smoothness of the video and the ability of the video to maintain the lines in the same position on the screen. This would be much more useful if this scheme was implemented in a UAV which could follow the runway or road. In this case, a small standard deviation would indicate that the control system was able to hold the image in the same area on the screen, meaning the controller would be working very well. If the standard deviation was larger, that would mean the controller had the tendency to bounce around the desired lines and was not able to maintain the exact heading all of the time. A screen shot of the result values display section is shown on the following page in Figure 4.32.


Results

Number of Times Rho Flag was Triggered

0 / 0

Number of Times Theta Flag was Triggered

0 / 0

Number of Times Main Flag was Triggered

0 / 0

Rho #1		Rho #2		Rho #3
0	Mean	0	Mean	0
0	Std. Dev.	0	Std. Dev.	0
1	Unused	1	Unused	1

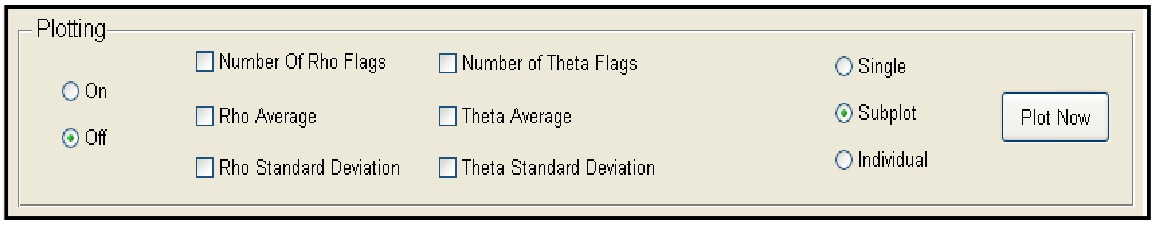
Theta #1		Theta #2		Theta #3
0	Mean	0	Mean	0
0	Std. Dev.	0	Std. Dev.	0
1	Unused	1	Unused	1

Figure 4.32: Runway Detection – GUI Result Values Display Section

4.3.3.3 Trend Plotting Section

The trend plotting section is very useful when trying to tune the values of the scheme in order to make the scheme run more efficiently instead of relying on the rho/theta correction subsystem block. This section allows a very versatile plotting routine to occur by using the check boxes to indicate which things the user wants to plot and then using another button to execute the plotting routine. The plotting can be further controlled by the on and off buttons. If the plotting is off, the plots will not be made until the ‘Plot Now’ button is pressed. If the plotting is turned on, the plotting routine will be executed upon completion of the simulation. The flag data, average values, and standard deviation from the result values display section can be plotted against the number of runs that have been performed. This data is constantly saved along with the data set used as input parameters and the run number. This allows a trend to be developed where a value in the input section is changed and then the results for all saved runs can be plotted such that a comparison can be made. This allows the user to easily decide if the change that was made affected the performance in a good way or in a bad way.

There are also options governing which method of plotting will occur. The user can select the plots to be made in a single plot per figure fashion, all in a subplot fashion, or in a grouped fashion where the lines from each group of parameters to be plotted are grouped together. This allows multiple methods of comparisons to be made either within a certain group or across groups to allow the user to examine one plot and see how the change affected more than one result value at the same time. The trend plotting section is shown below in Figure 4.33.


Plotting

On [✓ Off]

☐ Number Of Rho Flags
☐ Rho Average
☐ Rho Standard Deviation

☐ Number of Theta Flags
☐ Theta Average
☐ Theta Standard Deviation

Single [✓ Subplot] Individual

Figure 4.33: Runway Detection – GUI Trend Plotting Section

4.3.3.4 Video Analysis Windows

The main method of evaluating the performance of the runway detection scheme is working is by visual verification. This is done by examining several video output windows during the course of a simulation. These windows show the user how well the image pre-processing is working, how much of the image the scheme is actually using (the confinement), how well the edge detection is working, the strength of the peaks in the Hough space, and the actual placement of the lines resulting from the Hough peaks.

When a user runs a simulation, four video viewing windows and one Hough space window appears. The first window to appear simply shows the input image. The second window shows the confined image. This window is useful when perhaps the software is ‘losing’ one of the desired lines intermittently. This could be caused by the confined image being too small, so this window allows the user to watch the particular detail that is desired to be found and try to correlate it with an event in the window, like the detail moving out of the confined image area. If this happens, the user knows the confinement window is too small and an adjustment is needed.

The third window to open up displays the confined image after all pre-processing has been performed. What is seen here is the binary edge image after the morphological opening has been performed. This is when the user can really begin to detect the strong presence of the desired lines in the image. The desired lines are now fully filtered out and are obviously the most prominent thing on the display. If this is not true, then some adjustments may need to be made to one of several threshold values or the structuring element in the morphological opening operator. If there is a problem with the Hough transform in finding the wrong lines, this is the display where the root of the problem will be seen.

The fourth window is simply the output image from the image reconstruction block. This image is the confined image with the top and bottom sections rejoined to it with the Hough lines drawn in an overlay fashion onto the original image. This is the main output and this is where the visual verification of the functioning of the entire scheme is evaluated. If the scheme is having a problem and not finding the correct desired lines in the image, this display will show the line that it did find. Then, the user must decide how to filter out the line or line artifacts that the Hough transform is seeing that are obviously stronger than the line that is desired.

The fifth window is the Hough space display. This display shows all of the sinusoids created by the Hough transform of the edge image shown in the third video window. It is also easy to find the strongest Hough peaks by eye most of the time. This is another place where the user can determine why the scheme is not performing the way it is desired to. The Hough space can also help the user tune the Hough peaks neighborhood and the Hough peaks threshold value by looking at the strength and distribution of the sinusoids making up the peak. If the sinusoids do not form a direct peak, but are spread out over many pixels, then the neighborhood would need to be made larger to encompass the entire peak and the threshold would need to be made lower because the peak in this case would not be a peak, it would be a ‘hill’. The Hough space can also show an emergence of a peak which is not desired and can allow the user to determine exactly at what time this occurs. The user can then look at the other analysis windows to determine why this is happening and to try to rectify the situation in some manner, hopefully by a simple adjustment of the input parameters in the GUI. A screenshot of the analysis window cluster is shown below in Figure 4.34.

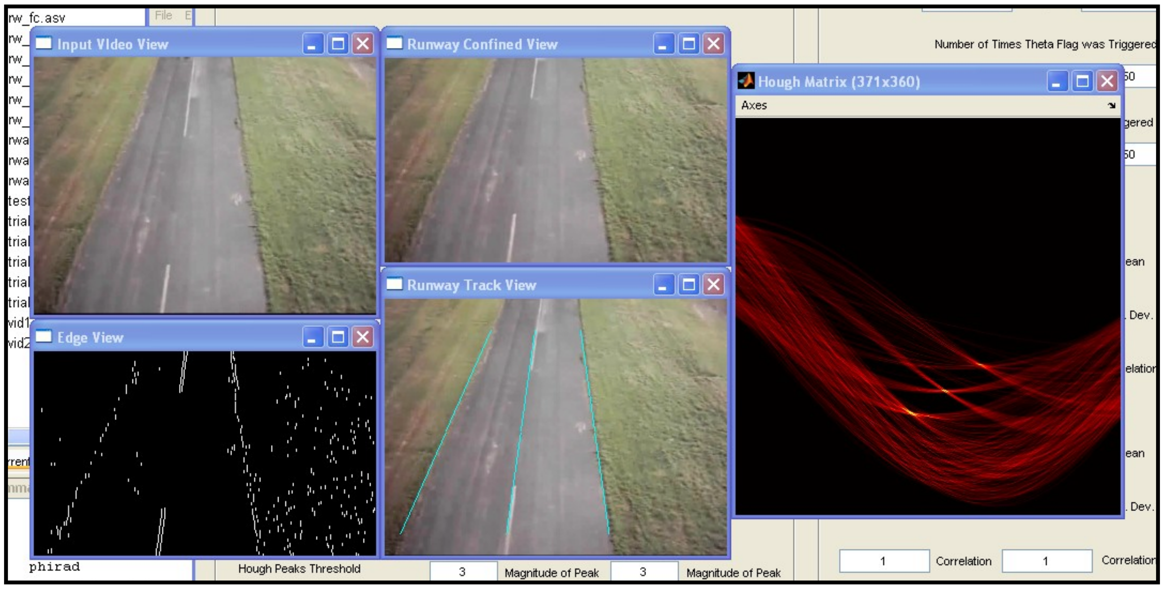


Figure 4.34: Runway Detection – Analysis Window Cluster

Chapter 5

Simulation Results and Discussion

5.1 Marker Detection and Tracking Results

The solutions to the Marker Detection and Tracking Problem can be evaluated in a number of ways. For this research effort, the different solutions were put through a series of tests which highlight their performance in computational workload, repeatability, robustness, and overall performance. These tests were conducted using pre-recorded videos created in the lab using the tanker and camera apparatus. The use of the pre-recorded videos allow a more fair comparison to be made, ensuring that one method is not encountering a video with more or less motion that the other method used. These videos were of a 10 second duration and they were recorded at a frame rate of 15 frames per second (FPS). By using the pre-recorded videos some computational workload is reduced by not introducing the use of the frame grabber and camera. The use of the frame grabber with Matlab® or Simulink® introduces a delay in the scheme because the software is trying to access the hardware through a Windows® based system. If the software were executed in any other platform than Windows®, it is thought that the frame grabber performance would be much better. Thus, eliminating the frame grabber from this simulation allowed a more accurate estimate of the computational workload to be made. The results of these experiments will be discussed in detail in the following sections: Section 5.1.1 covers the computational workload comparisons, Section 5.1.2 covers the estimation error comparisons, and Section 5.1.3 covers the robustness to noise comparisons. On the following page, Table 5.1, illustrates the array of simulations that was used for evaluation of the Marker Detection and Tracking Algorithm.

Table 5.1: Marker Detection – Breakdown of Trials Used for Evaluation

Computational Workload Comparison	
Frame-by-frame comparison	Matlab Profiler comparison
Trial 1	Trial 1
Trial 2	Trial 2
Trial 3	Trial 3
Trial 4	Trial 4

Estimation Error		
Actual vs. Estimate position	RMS position error	Roll Angle Measurement
Trial 1	Trial 1	Trial 1
Trial 2	Trial 2	Trial 2
Trial 3	Trial 3	Trial 3
Trial 4	Trial 4	Trial 4

Robustness to Vibration		
Roll Angle = 0	Roll Angle = 20	Roll Angle = 50
Vibration 1	Vibration 1	Vibration 1
Vibration 2	Vibration 2	Vibration 2
Vibration 3	Vibration 3	Vibration 3
Vibration 4	Vibration 4	Vibration 4

5.1.1 Computational Workload Comparison via Timing Data

In order to evaluate how efficient each of the three methods of marker detection are, comparisons were made between the methods using two different sets of data. The first set of data was obtained by using the Matlab® Profiler. In using the profiler, a list of each function that was used was given along with the total duration of time it took to execute and the number of times it was called. A list of the code was also given with times associated with each line number indicating how long the computer took to execute that particular line and how many times that line was executed. This data was then used to break down the code into sections that could be compared between each method. The code was broken down into seven sections and these sections are detailed in the following list:

1. *Reading the AVI* – when the computer reads the audio video interlace (AVI) file into memory from the disk.
2. *Pre-processing* – self explanatory – consists of the pre-processing steps in the code.
3. *Image Scanning* – fully scanning the image for white pixels.

4. *Scanning for Estimation* – scanning the 'search area' determined from the position estimation part of the code.
5. *Centroid Calculation* – calculation of the centroid of the markers in the images.
6. *Marker Definition* – determining which markers belong to each respective location on the plane, i.e. left wingtip, left stabilizer, etc.
7. *Other Lines and Overhead* – all other functions within the code such as matrix manipulation, etc. Each of these lines amounted to less than 0.01 seconds each.

The table on the following page, Table 5.2, lists the time spent on each of the previously described sections and compares them across the three different methods of marker detection. The table also lists the number of times each section was executed for this particular run. The data in this table was found while using the video file for Trial 1.

Table 5.2: Timing Comparison Between Marker Detection and Tracking Methods

Marker Detection/Tracking Speed Comparison using *Matlab Profiler*	**Modified Detection Algorithm (No Estimation)**		**Detection and Tracking Algorithm (Velocity Only)**		**Detection and Tracking Algorithm (Full Estimation)**	
Machine Vision Process	**Time (s)**	**Executions**	**Time (s)**	**Executions**	**Time (s)**	**Executions**
Reading the AVI	5.516	1	6.109	1	6.828	1
Pre-processing	4.422	150	4.779	150	5.34	150
Image Scanning	44.955	46,080,000	0.92	921,600	0.89	921,600
Scanning for Estimation	N/A	N/A	0.49	324,135	0.51	324,135
Centroid Calculation	0.4	15,328	0.15	24,870	0.22	24,870
Marker Definition	0.2	150	<0.01	3	<0.01	3
Other Lines and Overhead	0.867	N/A	4.376	N/A	1.79	N/A
Total Time	56.36	1	17.094	1	15.828	1

By looking at the previous table, several comparisons can be made. First, the time taken by reading in the AVI file is large because the AVI file is approximately 150 MB and there are some differences between the times it took to do this task and these can be attributed to the different background processes running in the Windows® environment. While looking at the pre-processing times, the same deduction can be made. There are slight differences here, but these must also be attributed to background processes. It is felt that since the pre-processing functions and the reading of the AVI are exactly the same in each version of code, that the indifferences must be attributed to an external source.

The largest difference can be seen in the image scanning section. In the Modified K-Means Detection Algorithm, each image is fully scanned resulting in over 46 million iterations for 150 frames of video. In contrast, the Advanced K-Means Detection and Tracking Algorithms only perform this on 3 frames resulting in just under 1 million iterations. The time is not any different on a per frame basis, but the fact that the scanning of the remaining frames is replaced by scanning smaller 'search areas' in the Advanced K-Means Detection and Tracking Algorithm reflect greatly on the increased efficiency of this algorithm. It can also be seen that while the Image Scanning section in the Modified K-Means Detection Algorithm took almost 45 seconds to complete, the two processes that comprise the same function in the Advanced K-Means Detection and Tracking Algorithm, Image Scanning and Scanning for Estimation, only take approximately one and a half seconds. This is equivalent to a 3000% decrease in the time spent on this section.

Te centroid calculation section of the code is not a computational intensive part but there is an odd phenomenon shown in the table. The centroid calculation section in the Modified K-Means Detection Algorithm was looped just over 15,000 times and took 0.40 seconds to execute while the same calculation was looped almost 25,000 times and took around half the time. At this time, there is no explanation for this phenomenon, but it is felt that this inconsistency is related to the background processes occurring at the same time and causing this time difference to occur.

One last thing to note in the table is the difference in the time taken by the marker definition section. This section is not a process that takes a lot of time but, the fact that it was looped for every frame, 150 times, in the Modified K-Means Detection Algorithm and in the Advanced K-Means Detection and Tracking Algorithm, it was only looped 3 times is very relevant. This relates back to the tracking part of the algorithm and shows that since the tracking part is actually tracking the markers and their names, then it only needs to run during the first three frames, instead of every frame. This reflects in favor of the efficiency of the Advanced K-Means Detection and Tracking Algorithm once again. Finally, the total time taken by each algorithm is shown and it can be seen that there is a significant difference, even in the Advanced K-Means Detection and Tracking Algorithm with the velocity only estimation and the full estimation. Using these results, the Advanced K-Means Detection and Tracking Algorithm is certainly the most efficient.

The second method used to evaluate the efficiency of the algorithms was by the use of the tictoc command in Matlab®. This command was used to calculate the exact time it took to process each frame of video data. This data was recorded and plotted for each video trial that was ran. The frame processing time for each method was plotted for every trial and compared. This can be seen on the following page in Figure 5.1. It can be seen from this figure that the data is very consistent and there is a clear trend that develops. It is very obvious that the Modified K-Means Detection Algorithm is the slowest and this was also shown in Table 5.2. This algorithm averaged a 0.085 second processing time for each frame of video. This is equivalent to a frame rate of approximately 11 Hertz (Hz). There is, however, a drastic drop when looking at the methods which use estimation to find the markers. The Advanced K-Means Detection and Tracking Algorithm which uses velocity only for estimation is much faster, averaging just under 0.05 seconds per frame of video which is equivalent to approximately 20 Hz. The Advanced K-Means Detection and Tracking Algorithm which uses velocity and acceleration for marker position estimation is even faster, averaging less than 0.03 seconds per frame. This is an equivalent processing speed of approximately 33 Hz. Something that should be noted again is that the inconsistencies seen in the frame processing time data in Figure 5.1 reflect some background processes that are interfering with the smooth operation of the code. In only one instance, in Trial 4 using the Full Estimation code, did the entire simulation run without one major interruption from the operating system. This is reflected in Figure 5.1 in the lower right hand plot for Trial 4. By looking at the plots below and referring to the table above, it is obvious that the best performing algorithm is the Advanced K-Means Detection and Tracking Algorithm using the full estimation.

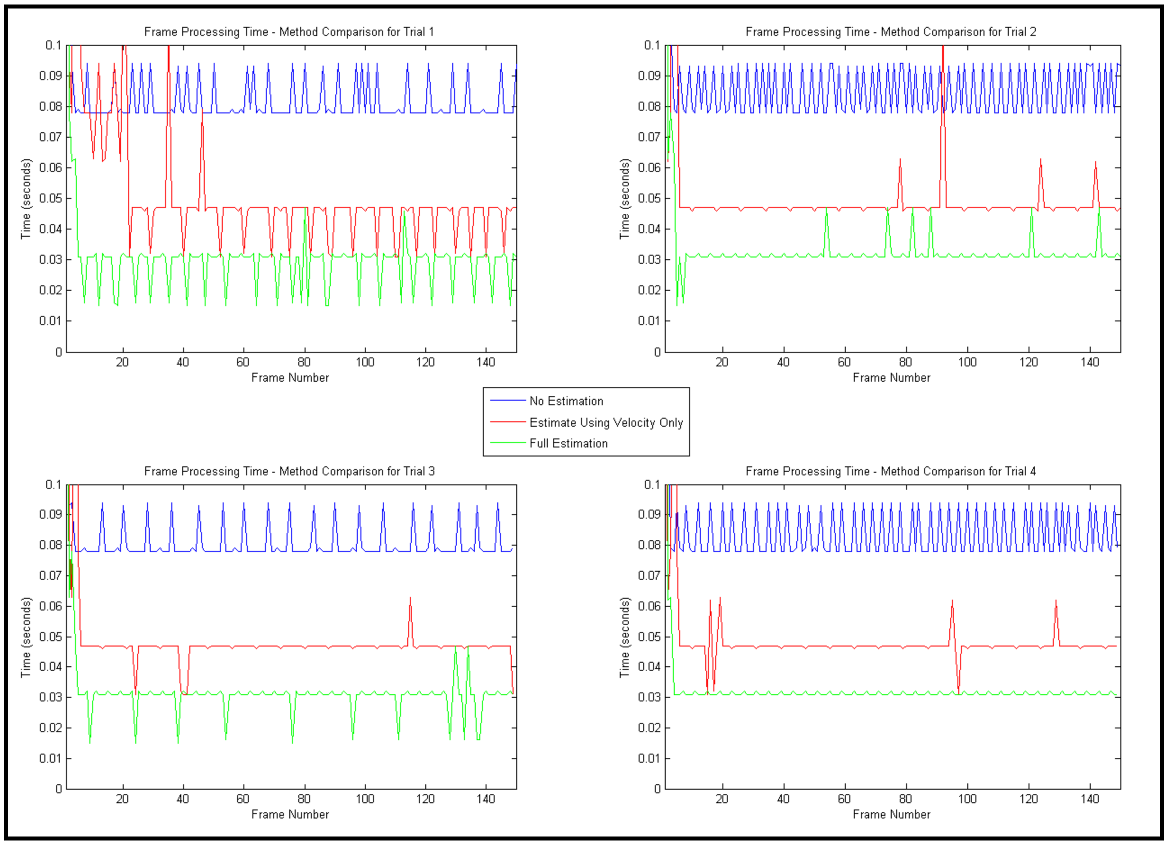


Figure 5.1: Frame Processing Time Comparison Between Methods for All Trials

In examining the plots above, a noticeable transient occurs that should be examined closer. The transient is a direct result of performing a complete image scan during the first three frames that the Advanced K-Means Detection and Tracking Algorithm processes. This transient is also present in the Modified K-Means Detection Algorithm but it is only present in Trial 2 and Trial 4. Since this transient is not shown 100% of the time and this code does not change the way it scans the image after 3 frames of video like the codes using estimation do, it is thought that this is somehow related to the initial allocation of memory for all of the images. If this is true, then it also means that the large transient seen in the codes which use estimation is not created entirely by the transition from full frame scanning to estimation but the memory allocation must also have some effect on these codes as well. This transient is about 3 times the average value of a frame processing time for the code using the full estimation. This is shown below in Figure 5.2. In this figure, the transient can be seen until the third frame, at that time, the frame processing time drops to the normal value seen for the rest of simulation. In support of the theory above, a noticeable difference can be seen in the transient before frame two and after frame two. It is thought that the transient before frame two is due to the memory allocation coupled with the full image scanning and the transient after frame two is the pure difference between scanning the entire image and not scanning the entire image. This is further supported by seeing that in frame two the frame processing time is approximately the same value as the Modified K-Means Detection Algorithm frame processing time. This is a direct reflection that in the first three frames, all of the methods are performing the exact same task.

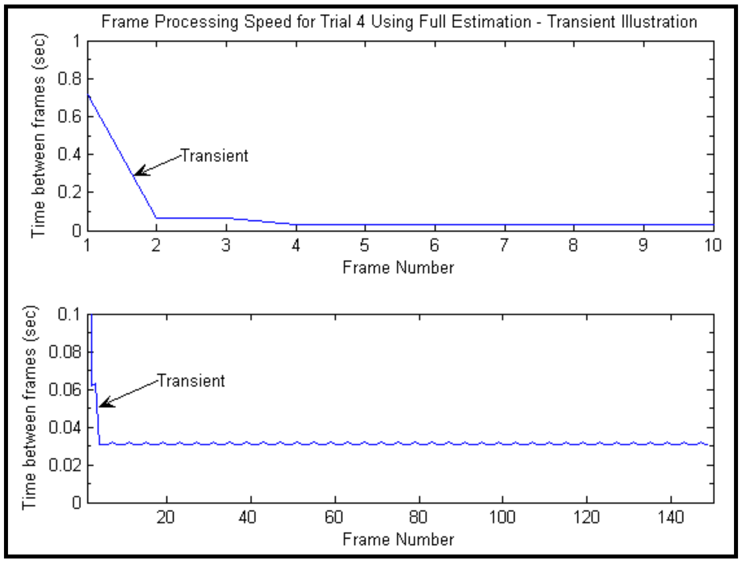


Figure 5.2: Transient Illustration from Frame Processing Speed Plot for Trial 4

It was mentioned in Chapter 4 that a point was made regarding the accommodation of the loss or gain of markers during a refueling operation. This issue was addressed with a separate software method utilizing a while loop and an unknown number of markers instead of a for loop with a known number of markers. Similar to the other methods, this method was also evaluated for its computational efficiency. Since the tracking part of the algorithm was not implemented in the improved version of the software, a fair comparison between those methods can not be performed. Therefore, the only fair comparison that can be made is between the original Modified K-Means Clustering Algorithm and the improved version of the same software. This was performed with the tictoc command in Matlab®. The comparison between these two methods and the results from the methods using estimation is shown below in Figure 5.3. The methods using estimation are shown only for reference.

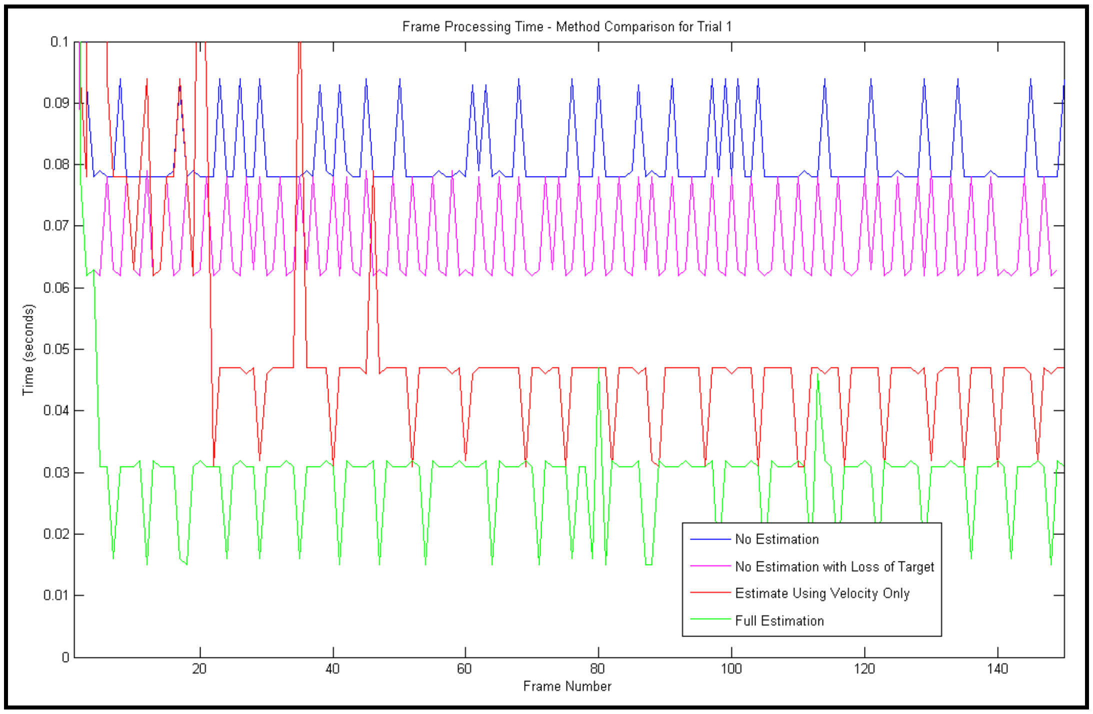


Figure 5.3: Frame Processing Time Comparison Of New vs. Original Algorithm

In examining Figure 5.3, the addition of the method which compensates for the 'Loss of Marker' is evident with the magenta line. It can be determined from this plot that the software which uses the while loop is slightly more efficient than the original software using the set number of for loops. The average time per frame using the original software is approximately 0.085 seconds while the average time per frame using the software which accommodates for the loss of marker visibility is approximately 0.07 seconds. This reflects an approximate 18% decrease in the average time per frame. This same speed increase evident here in the comparison of the two Modified K-Means Clustering Algorithms would not be directly applied to the methods using estimation if the same while loop is implemented in those versions. This is because the versions performing estimation only use this method of complete image scanning during the initial three frames. Therefore, the 0.015 second increase in speed could only be applied to the initial three frames and the frame processing time seen in Figure 5.2 and Figure 5.3 regarding the speed of the methods which use estimation would still be valid for the time after frame 3.

In fact, the implementation of the while loop to accommodate for the loss of marker visibility would be more difficult to couple with the estimation part of the code due to the fact that the estimation constantly uses information from the last three frames. Therefore, to accommodate the disappearance of a marker additional software would have to be written to ensure that the position of the marker is still estimated and tracked using either the last known inertial information or current inertial information from another marker in the array. Also, the search area size would probably have to be automatically increased if a marker was lost to ensure that it could be found again due to the erroneous estimations that would be made regarding its location, if and when it reappeared. Since the point of the loss of marker visibility was made after this research was complete it was not within the scope of this effort to accommodate for the loss of marker visibility in the more complicated method using estimation. The addition of the code which accommodates for the loss of marker visibility was intended to demonstrate that the assumption regarding the fixed number of markers could be removed but for it to be applied globally to all versions of the K-Means Clustering Algorithm would require a structural overhaul to deal with this dynamic situation.

5.1.2 Estimation Error

The estimation error comparison is an important part of the performance evaluation of these software methods. If there were times during a simulation when a method would have a false indication, this would be a great point to start evaluating each method. Since this is not the case and each method does its job of detecting the markers very well, one must find other avenues to measure their performance in this respect. The first way is to examine the actual positions and the estimated positions. Figure 5.4, on the following page, shows the estimated position vs. the actual position for all four trials using the full estimation software. The estimated positions are shown by the red line and the actual positions are shown by the blue line. The green circles indicate the position of the markers at the start of the video and the red circles indicate the position of the markers at the end of the video. As indicated in the plots, the differences are very minute and as a result of this, it is very difficult to indicate the performance in this manner since comparisons are hard to make when the two lines being compared overlap so much.

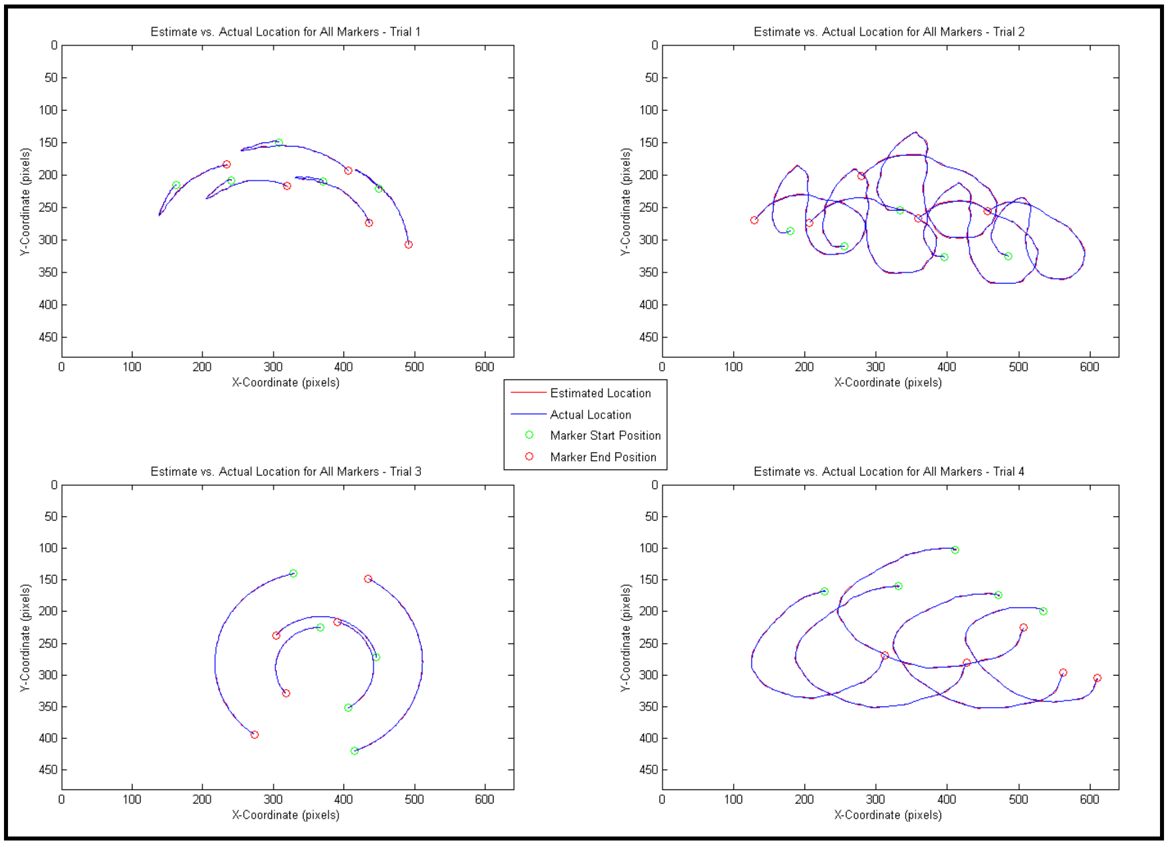


Figure 5.4: Estimated vs. Actual Position for All Trials

Another representation of the estimation error can be seen in Figure 5.5, on the following page. The estimated positions are shown by the red line and the actual positions are shown by the blue line. This figure illustrates the actual position versus the estimated position in terms of X and Y coordinates in separate plots. In this example, it is easy to see how the estimated position constantly overshoots the actual position but, in all cases, this overshoot is on the order of less than 2 pixels which is negligible. This plot was taken from Trial 2 simulations where the movement was very erratic and unpredictable. Figure 5.6, also on the following page illustrates the same comparison between the actual position versus the estimated position but, this plot is selected during the section of the Trial 1 simulation where the tanker comes to a stop and changes directions and the X and Y coordinates are plotted against each other. This plot illustrates the constant overshooting problem very well.

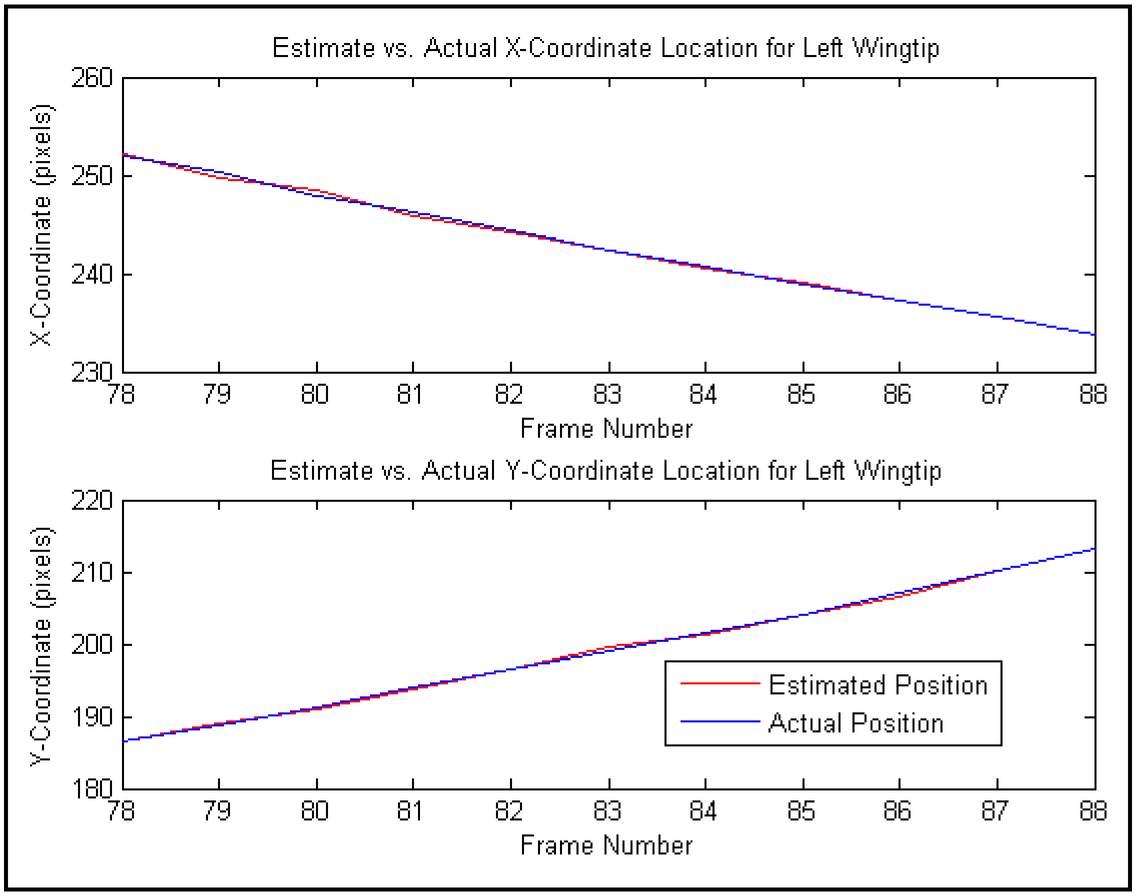


Figure 5.5: Estimated vs. Actual Coordinates for Left Wingtip in Trial 3

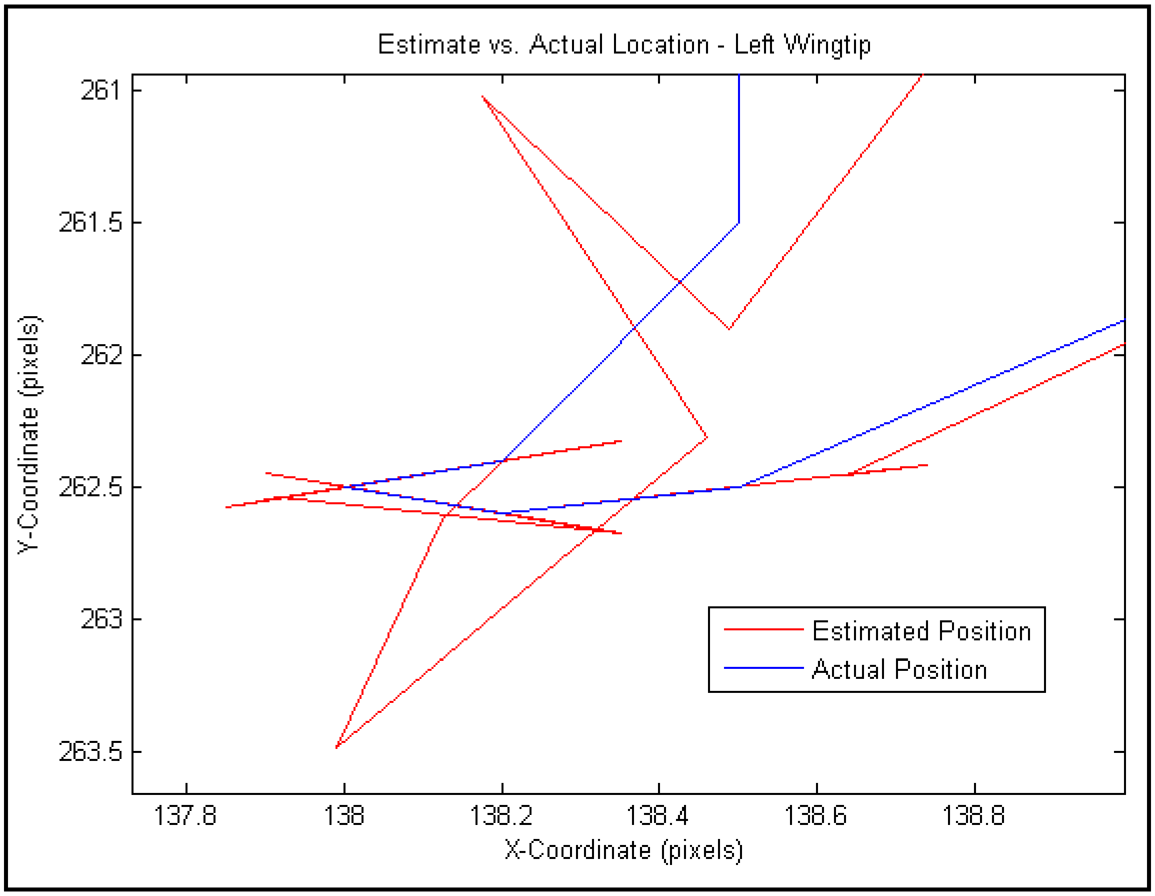


Figure 5.6: Estimated vs. Actual Position for Left Wingtip in Trial 1

Another example of the estimation error is shown on the following page in Figure 5.7. Figure 5.7 illustrates the distance error from the actual position to the estimated position. It can be seen from these plots that the error is below 2 pixels for all markers in almost every instance. This is a good indication that the estimation is working well but the overshoot seen in earlier plots like Figure 5.5 and Figure 5.6 indicate that the estimate is overshooting the actual position very frequently. The plots presented here were all produced using the full estimation software but similar results can be seen in plots produced using the velocity only estimation as well. This phenomenon will be explained further in the RMS error plots on the following pages.

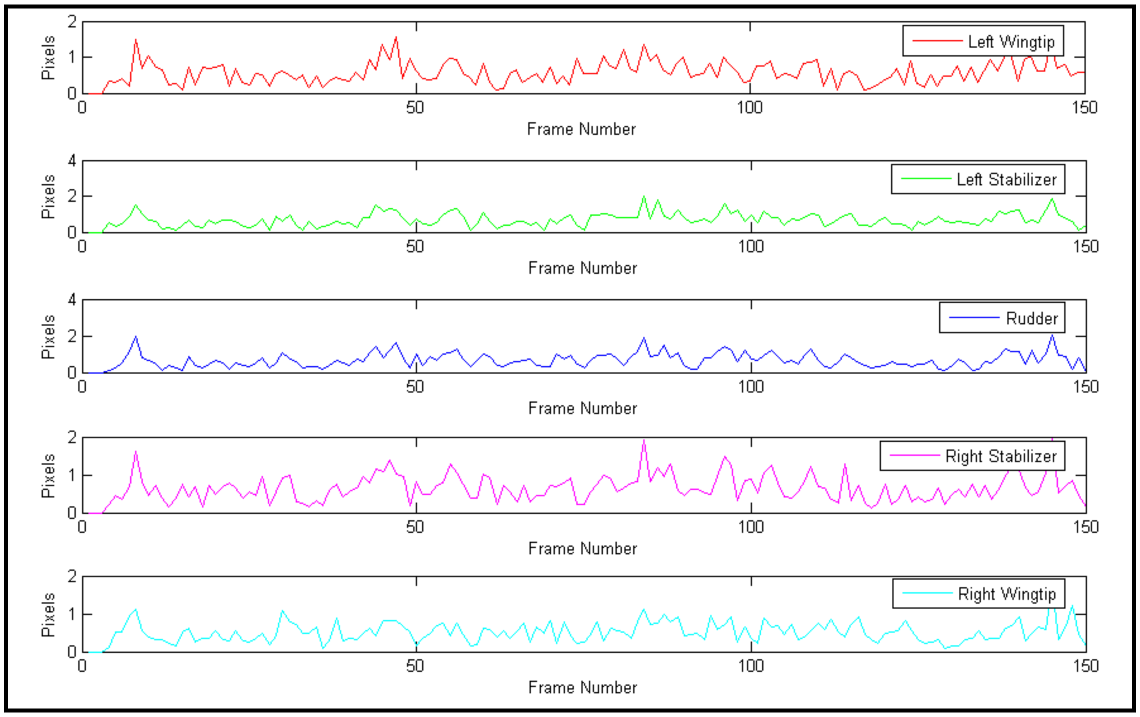


Figure 5.7: Marker Position Error Calculated Using Absolute Distance

Although the plots above and on the previous pages are the result of a single detection method (full estimation), similar plots were examined for the other methods as well. The plots all indicated the same minute differences. As a result of this, the roll angle measurement, which was calculated from the wingtip marker positions, was also examined for 'dropouts' and since there were no false indications in the actual positions, there were similarly no false indications on the roll angle measurement plots. These plots can be seen on the following page in Figure 5.8.

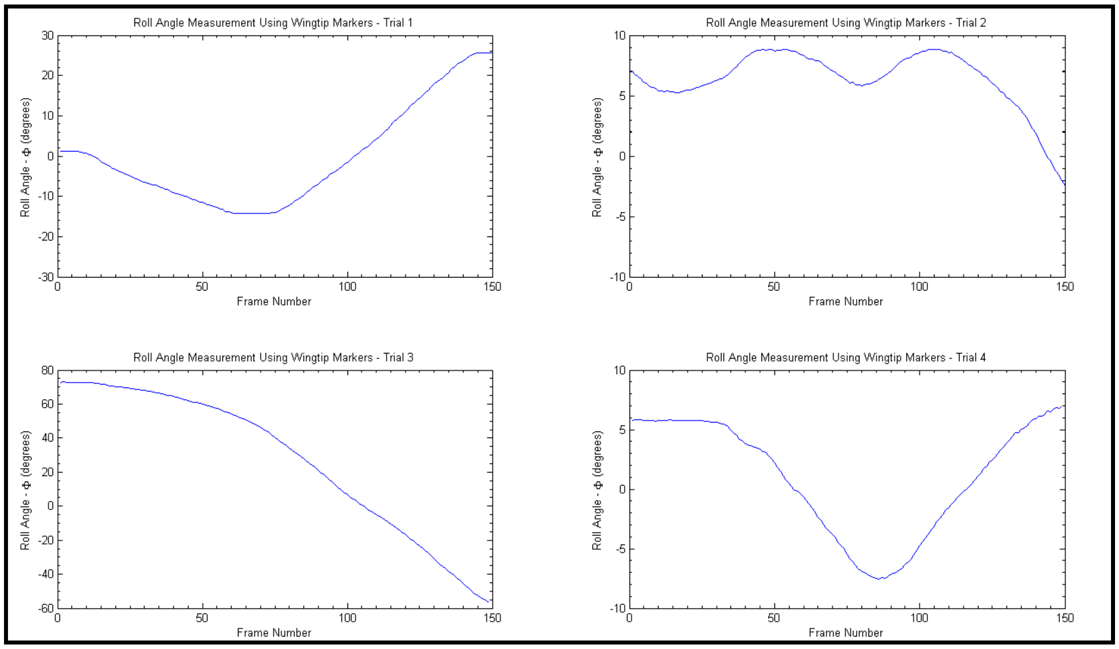


Figure 5.8: Roll Angle Measurement from Marker Detection Software for All Trials

By close examination of Figure 5.3, Figure 5.4, and Figure 5.7, it could be said that all of the Marker Detection software works very well. In order to quantify this indication, the root mean square (RMS) position error was calculated and compared. The RMS position estimation error was calculated for the Advanced K-Means Detection and Tracking Algorithms only. This is due to the fact that the Modified K-Means Detection Algorithm does not perform an estimation and therefore, does not have an error to be calculated. The RMS position error was calculated for all markers, for both methods, and for all four simulation trials. Table 5.3, on the following page, shows the RMS errors calculated from those trials.

Table 5.3: RMS Position Estimation Error – Method Comparison for All Trials

Trial #	Estimation Type	Left Wing	Left Stabilizer	Rudder	Right Stabilizer	Right Wing
1	Velocity Only	0.8306	0.4556	0.3829	0.378	0.5478
	Full Estimation	0.7341	0.4500	0.4725	0.4414	0.5621
2	Velocity Only	0.6145	0.6570	0.6486	0.6243	0.5398
	Full Estimation	0.6479	0.7684	0.7643	0.7363	0.5759
3	Velocity Only	0.4310	0.4082	0.3546	0.4519	0.3590
	Full Estimation	0.4987	0.3901	0.3546	0.5299	0.3804
4	Velocity Only	0.4937	0.5285	0.5434	0.4916	0.4762
	Full Estimation	0.4887	0.5776	0.5969	0.5404	0.4660

In examining Table 5.3, two things will become obvious. The first thing is that both of the software methods that perform estimation work very well. In fact, in the 40 RMS error calculations performed, not one value was greater than 1 pixel of RMS error. This is also evident in Figure 5.9, below. Figure 5.9 is a plot which compares the results from the table above and it is broken down into four plots, one for each trial.

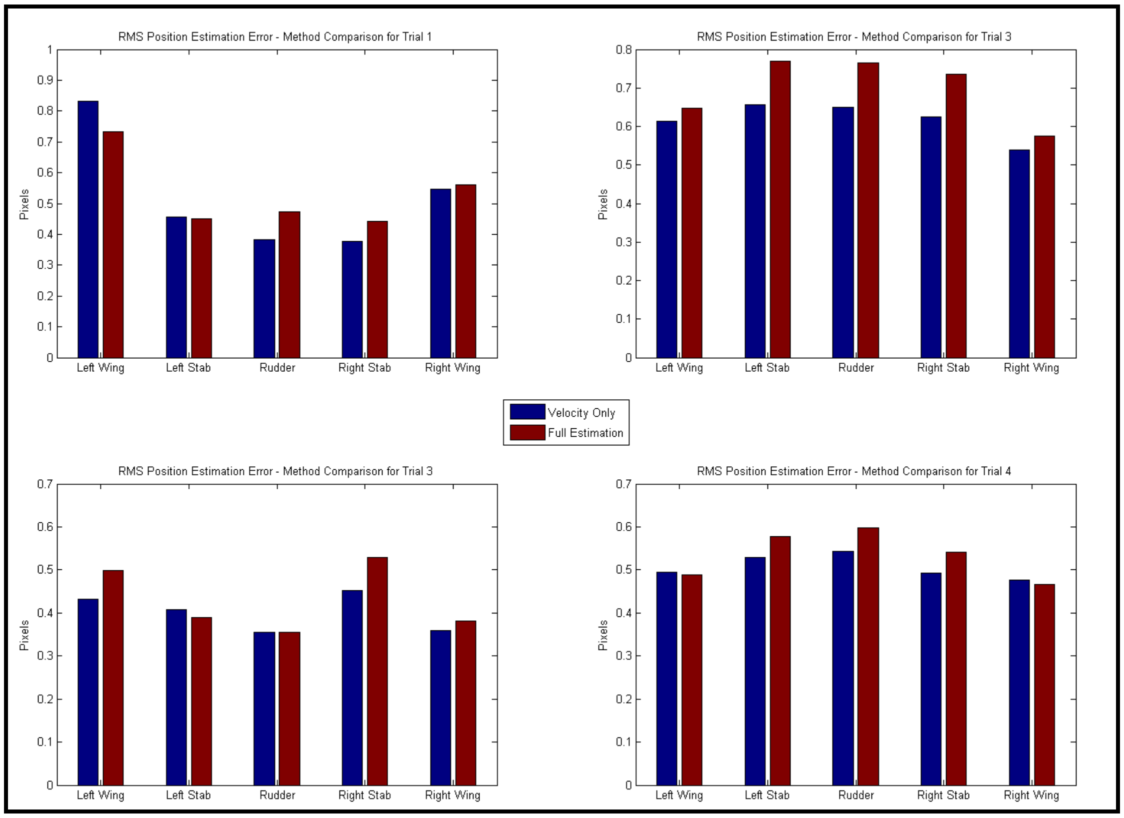


Figure 5.9: RMS Position Estimation Error – Method Comparison for All Trials

The second thing that will become obvious is that there is no clear winner in these results. For one method to be declared better performing than the other, the RMS error differences would have to be clear cut. In the trials outlined above, the method using only velocity for estimation only has better RMS errors roughly 50% of the time and vice-versa. In examining Figure 5.9 for Trial 1 for example, it can be seen that for the left most two markers, the full estimation method outperforms the velocity only method. But, in looking at the other three markers, it is clear that the velocity only method outperforms the full estimation method. This type of split is again seen in Trial 3 and Trial 4. In Trial 2, the clear winner is the velocity only estimation method.

There is some indication as to what exactly causes this split. In examining the types of videos used in the trials there is only one clear difference that can be related between the movement in the video and the results of the RMS errors and that is, in fact, in Trial 2. Referring to Figure 5.4 on a previous page, the video for Trial 2 could be described as quite erratic and very unpredictable. It is again, in this trial, that the RMS error for the velocity only estimation method indicates that it outperforms the full estimation method. Therefore, the theory is that when there is sufficient motion, the full estimation using the acceleration calculation is actually over shooting the actual position much more often and causing a larger RMS error. This relates to a fact stated earlier that if the frame rate is sufficient enough to capture the motion, the full estimation would work better in theory, but if either the frame rate was too slow or the motion too high then the estimation would miss the actual positions more. It appears as though this is exactly what is happening in Trial 2 and partially in the other trials as well. But, even under these conditions, both methods still perform very well.

5.1.3 Robustness to Noise

A robustness to noise study was performed to assess the performance of the software to a noise source such as vibrations. Vibrations can not be considered a complete and exhaustive study of image noise but within the scope of this research, vibrations were determined to be suitable for the robustness to noise study. Vibrations were chosen because they are fairly easy to create and they could be measured and quantified with equipment already on hand. Vibrations are also a very practical disturbance that may be encountered in a machine vision situation such as this one in an UAV.

In order to create the vibrations, a motor with an off center weight was attached to the top of the camera tripod. The motor was then connected to a variable power supply which allowed varying speeds of the motor. A Crossbow IMU was used to measure the vibrations applied to the camera and the data was recorded for analysis. This entire setup was described in detail in Section 4.2.1.5. Once determinations were made as to the exact desired vibration based on the visual movement of the camera, the related voltages being applied to the vibration motor were recorded so that the vibration could be recreated. Once the setup was complete, the desired vibrations were recorded and calculations were made to quantify the different vibrations. Once the vibrations were recorded, the accelerations in all three axis' were used in power spectral density (PSD) calculations. The PSD data allowed the frequency and amplitude of the vibrations to be determined. These values were used to quantify the vibrations used in the robustness tests. The information gathered from the PSD plots is shown in Table 5.4, below.

Table 5.4: Test Conditions Used for Robustness to Vibration Tests

Condition	Primary Frequency (Hertz)	Amplitude (dB/Hertz)	Harmonics (Hertz)
No Motion	39.4	2.77E-06	N/A
Vibration 1	11.06	0.005248	22.12, 33.11, 66.58
Vibration 2	14.21	0.01204	28.42, 64.01
Vibration 3	16.63	0.01378	8.31, 24.98, 33.22, 41.78, 49.88, 58.26, 66.54

By looking at the PSD plots, shown on the following page in Figure 5.10, it was easy to determine the primary frequency of vibration and all of the related harmonics. Interestingly enough, some harmonics were not present in some trials because of the damping effect of the legs of the tripod. Another fact gathered by using the PSD plots was that with the tripod totally still, there was a large peak at 39.4 Hz in the X-direction and a second smaller peak in the Y-direction. This could only be attributed to measurement noise created inside the IMU itself. The 39.4 Hz noise even appeared on the PSD plot for Vibration Trial #2 but was not noticeable in the other trials. The PSD plots were extremely useful in this case to help to verify that each vibration trial used was actually stronger than the previous one but not only amplitude information was gained. The frequency information was an added bonus and allowed a further delineation to be created between the vibration trials.

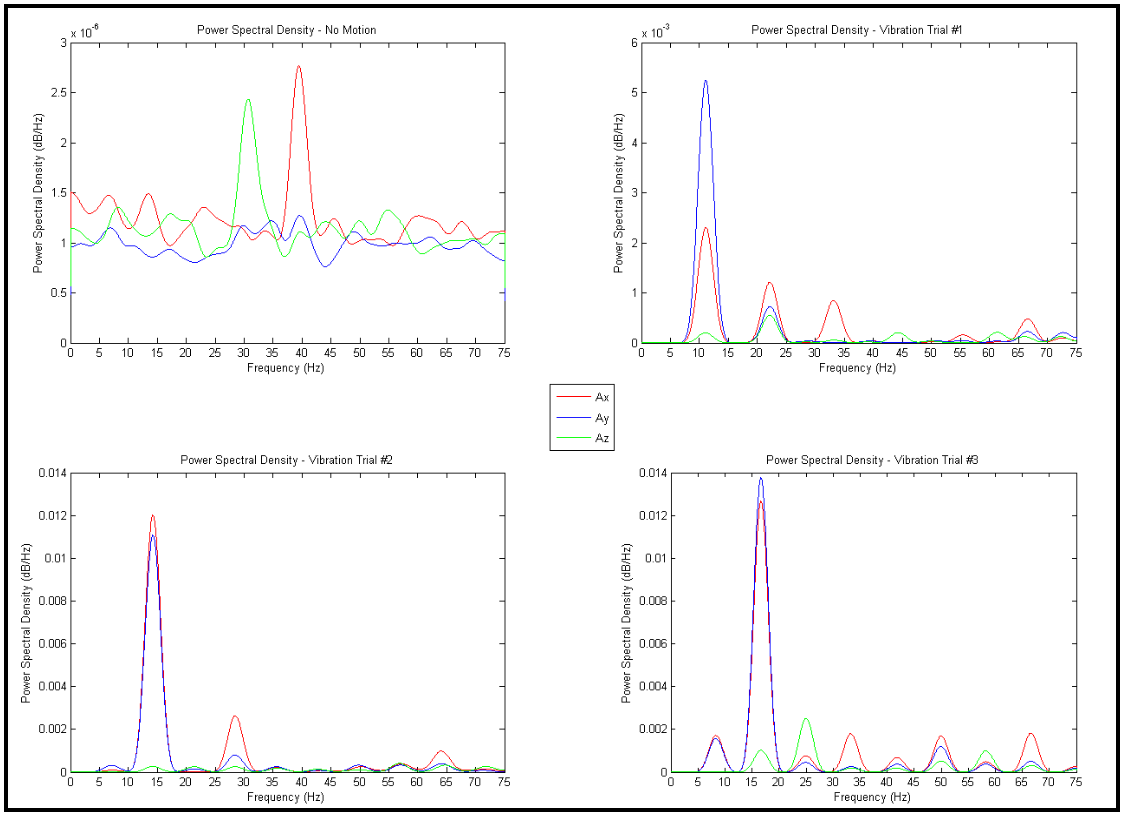


Figure 5.10: Power Spectral Density of Vibration Conditions

The vibration noise that was created for this study was most likely higher in frequency that would be encountered in a real situation. The amplitude of the vibrations, however, cause some great excitation in the measurements taken by the software which was the desired end result of the application of the vibrations to the camera mount. The fact that the vibrations also had numerous harmonics in different directions which would have intermittent waves of canceling each other out and opposing each other allowed a much more intense screening of the robustness of the software to take place than would have been achieved with a constant vibration only. The data taken by the software to analyze the robustness was based solely on the roll angle measurement. In these tests, the roll angle was set to three specific values for each vibration trial. For each of these 12 different sets of conditions, the roll angle was measured 60 times, once per second. Once this was complete the RMS error of the roll angle measurements for all of the conditions was calculated and compared. The table outlining the RMS errors is shown below in Table 5.5.

Table 5.5: RMS Error of Roll Angle Measurements for Vibration Tests

	RMS Error (deg)		
Condition	$\Phi = 0°$	$\Phi = 20°$	$\Phi = 50°$
No Motion	0.004941	0.003679	0.01315
Vibration 1	0.1063	0.056434	0.089607
Vibration 2	0.1971	0.2035	0.19598
Vibration 3	0.4215	0.2693	0.3303

As can be seen from the table, the roll angle measurement RMS error did increase as the vibration amplitude and frequency increased and there were no worrisome differences between the different roll angles for each vibration trial.. The RMS error did not, however, increase to an undesirable amount. The vibration presented to the camera in Vibration Trial #3 was certainly more than could be expected in a real situation and the software appeared to handle it without duress. Figure 5.11, on the following page, is a plot of the RMS errors for each of the vibration trials. The data appears fairly consistent with the exception of Vibration Trial #3. This trial exhibited an undue amount of vibrations to the camera which attributed to the slightly uncorrelated results.

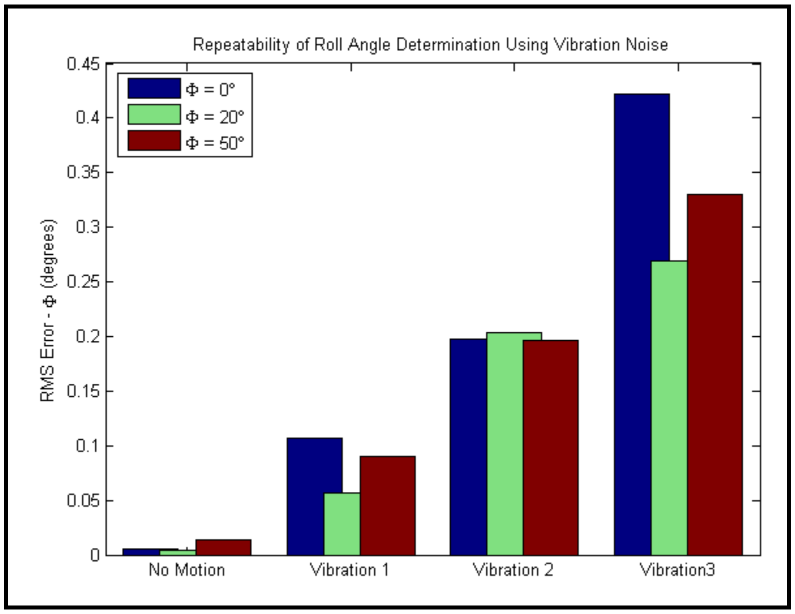


Figure 5.11: RMS Error of Roll Angle Measurement for Vibration Trials

5.2 Runway Detection Results

Normally, the assessment of the performance of the Runway Detection scheme would not be difficult but, because the video used in the Runway Detection scheme was taken from an aircraft that was not yet instrumented, there were no other data sets, such as GPS, associated with the flight that could be used for comparison. As a result of this, the performance evaluation is quite limited in the scope of tests that can be performed. This limitation does not exclude visual means of evaluation however, and that constitutes the majority of the performance evaluation of this scheme. The visual means of evaluation is comprised of actually looking at the output of the scheme and verifying that it is working and this is covered in Section 5.2.2. One other aspect of the scheme that was explored was the computational workload. This is always an important aspect of software when dealing with MV applications. Therefore, a full computational workload analysis was performed and it is detailed in Section 5.2.1.

It should be noted, however, that with the availability of flight data associated with the flight videos, the performance metrics would be easily defined. If flight data was available, the visual means of validation and the use of the computational workload calculations would still be used but, there would be additional things to consider as well.

First, with the availability GPS data for the flight, a small addendum to the Runway Detection scheme could be made that could output the GPS coordinates of the runway using the known position of the aircraft. This could then be used as a judge of the performance of the Runway Detection scheme. In order for this comparison between the estimated runway position and the actual runway position to be made, the actual runway position would have to be known. The actual position could be determined by using a static GPS unit and mapping the runway manually. This data could then be extrapolated and compared to the estimated position. Secondly, if a directional control system such as a heading hold controller could be implemented, the algorithm could actually be tested in it runway following ability and with this kind of experiment, the tracking error could be calculated which in this case, would be the best gauge of performance possible. Table 5.6, below, illustrates a breakdown of the tests performed on the runway detection scheme.

Table 5.6: Runway Detection – Breakdown of Trials Used for Evaluation

Computational Workload Comparison	**Visual Examination**
Simulink Profiler comparison	
Trial 1	Trial 1
Trial 2	Trial 2

5.2.1 Computational Workload Analysis

In order to evaluate the computational workload of the scheme, the Simulink Profiler was used. The profiler was able to break down the time spent on each block in the scheme and these blocks and their times were assembled in Table 5.7, on the following page. The description of the blocks can be found in Section 4.3.2, therefore no additional explanation will be given here.

Table 5.7: Timing Analysis of the Runway Detection Scheme

Speed Comparison using *Simulink Profiler*	**Runway Detection** *No Video Output*	
Machine Vision Process	**Time (s)**	**Executions**
Model Initialize	0.3906	1
Reading the AVI	1.6562	150
Pre-processing	3.4374	150
Hough Transform	2.4218	150
Rho/Theta Correction	0.09375	150
Image Regeneration	0.3125	150
Other Lines and Overhead	0.2658	N/A
Total Time	7.9062	1

The Runway Detection tests were performed using videos that were 5 seconds in duration and recorded at a frame rate of 30 FPS. This combination creates a video that is 150 frames long and this is reflected in the timing analysis data in the above table in the number of executions column. Since there were only 150 frames, there were only 150 executions of each block.

The timing analysis was broken down into the major subsystems and the model initialization function. It is easy to see that the pre-processing subsystem takes the most time compared to all of the other subsystems. It accounts for about 43% of the computational workload. This reflects on the importance of a bare minimum pre-processing scheme. The Hough transform is the next most computational intensive subsystem. Surprising is the fact that it did not exceed the time spent in the pre-processing subsystem as the number of calculations the Hough transform must perform for every frame is enormous.. At 2.4218 seconds, the Hough transform comprises 30% of the computational workload. The last large time consumer in the list is the reading of the AVI file function. This function, like in the Marker Detection software is another of the computationally intensive functions. It consumed 1.6562 seconds of the 7.9062 second total time, which amounts to about 21% of the total time. These two subsystems and one function amount for 94% of the total computational time required by this scheme. The rest of the time was spent on other smaller functions which individually comprise less than 0.5 seconds each but they amount to the other 6% of computational workload exhibited by the scheme. Since there are many other blocks in the scheme, in fact, almost 10 times the number of blocks that comprise 94% of the computational time, the time spent on all of these blocks is negligible.

In order to have a fair estimation of the real computational frame rate, it is necessary to discard the time taken by the model initialization function. The model initialization function only happens once in the simulation but for a frame by frame look at the computational time, it needs to be removed because it does not happen during frame processing, only before. With discarding this value, the ‘total time’ of processing stands at 7.5156 seconds. Using the total number of frames processed, which is 150, results in a frame rate of approximately 20 Hz. This frame rate is probably adequate to be used in an UAV for navigation and since the scheme was not written in a real-time environment the possibility exists for the speed of this scheme to be increased which could yield even better performance for a real-time application.

5.2.2 Performance Analysis

As mentioned before, the Runway Detection scheme is very difficult to analyze without the availability of video taken from an instrumented aircraft. Therefore, most of the performance evaluation is based on visual examination of the output only. The following figures show examples of the output of the Runway Detection scheme. The colored lines on the images indicate the position that the scheme has detected there to be a strong presence of a straight line. The strong presence of a straight line relates to the sides of the runway and the center line of the runway. It is evident in the figures presented below that the scheme is working to the best of its ability and it performing as it should, detecting the three most prominent lines on the image. Figures 5.12, 5.13, 5.14, and 5.15 are examples taken from real flight video and are presented for performance evaluation purposes. These figures an be found on the following pages.

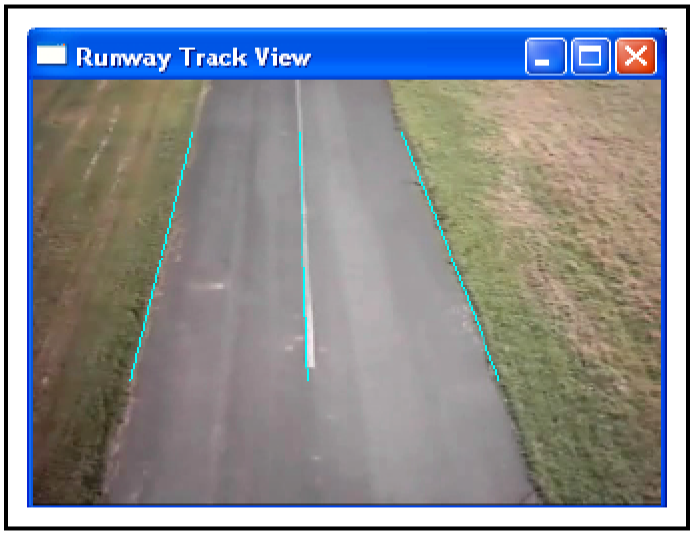


Figure 5.12: Performance Evaluation Image #1 for the Runway Detection Scheme

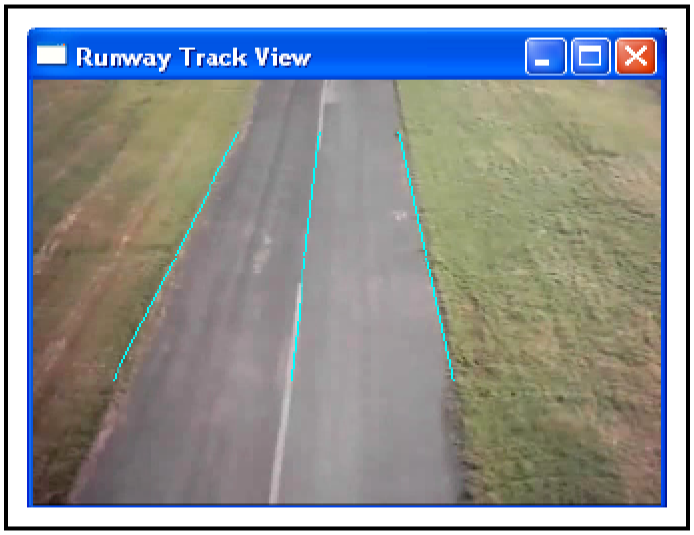


Figure 5.13: Performance Evaluation Image #2 for the Runway Detection Scheme

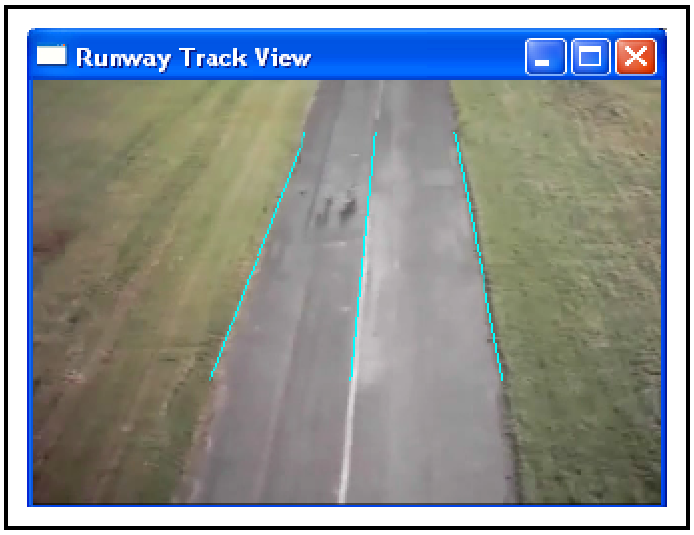


Figure 5.14: Performance Evaluation Image #3 for the Runway Detection Scheme

These figures show an almost unflawed performance in detecting the lines comprising the runway. Ultimately, an UAV with a control system would have no problem following this runway with the accuracy given by the Runway Detection scheme as long as the scheme was able to execute fast enough to accommodate the speed of the aircraft. The accuracy of this scheme is not where a failure would likely occur, it is in the speed of execution where the real problem with implementation lies. Figure 5.12 represents a perfect frame of detection. Figure 5.13, 5.14, and 5.15 all have some slight misjudgment of the actual edge of the runway or the center line. But, it is easy to see that the trajectory needed to follow this runway could easily be discerned from images displaying this type of accuracy.

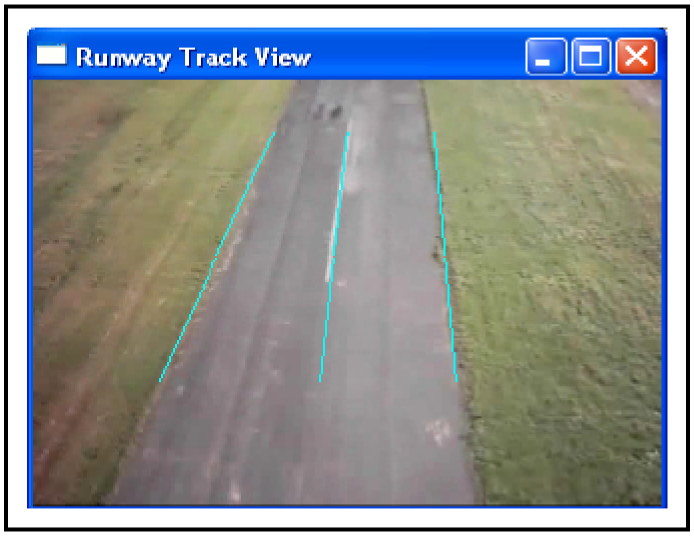


Figure 5.15: Performance Evaluation Image #4 for the Runway Detection Scheme

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The purpose of this research effort was to investigate the feasibility of MV applications in an UAV. These applications consisted of marker detection on a tanker aircraft for the purposes of AAR and runway detection for the purposes of following the trajectory of a runway, road, or pipeline. Through the research presented here, the objectives regarding these MV problems were satisfied. This research has shown that this type of MV application is feasible and it is assured that this type of technology will be applied in the future. The possibilities that extend from an UAV having these abilities are numerous and they will be invaluable to the future of military aviation.

The Marker Detection and Tracking software has been shown to work very well under numerous conditions simulated in the lab environment. Taking into account the data from all of the experiments, the Advanced K-Means Detection and Tracking software would be the clear favorite. This software continuously yielded a faster computational time than the other methods although the RMS errors were a toss up between the two methods using estimation. The software could be accelerated even further while also gaining robustness to the loss of marker visibility situation if the K-Means Clustering Algorithm using the while loop was implemented along with the tracking algorithm. This was detailed earlier in an effort to address the problem of loss of visibility of markers during a simulation. Regardless, this method showed solid performance in every aspect and would be the choice for further research in this area.

In every case shown, all of the software versions were able to continuously find the location of the markers on the aircraft. In the cases shown where estimation was involved, the estimation error was very negligible, with RMS errors being less than 1 pixel. This shows that a marker position estimation scheme could be relied upon under much more adverse conditions. An attempt at simulating these conditions was made by using vibrations to excite the motion of the camera. In these cases, the repeatability of the roll angle measurements was proven to be acceptable even under the most violent vibration activity. This was proven by showing that the RMS error of the roll angle measurement was consistently less than 1 degree. This is not an indication that the MV software could be used for roll angle measurement but it was an indication to the robustness of the marker detection algorithm. These tests indicated that the algorithm could still provide acceptable results using blurred markers caused by vibrations which could possibly be a concern for an small UAV. The computational loads exhibited by the marker detection software indicate that it could be used in a real-time environment. This fact only strengthens the claim that this is, in fact, a feasible operation. The frame rate achieved by the Marker Detection and Tracking software using full estimation was greater than 30 Hz. It is currently thought that a computational speed of 20 Hz or more is acceptable to ensure operational effectiveness within a UAV platform. Therefore, based on the information available about the performance of this algorithm such as the computational loads and the estimation errors, it is thought that it could be applied in an UAV. Furthermore, if the current growth rate of high speed, efficient, lightweight, compact computers and research efforts such as this one continues, the problem of AAR could very well be addressed in a real-life application in this decade.

The Runway Detection scheme was a difficult problem to address and the analysis of the results proved to be even more difficult. The concept of the use of runway detection for things such as automatic landing or simply following a trajectory is a very feasible idea as proven by this research effort. This effort proved that runway detection could be used in a real-life application because of the speed and effectiveness of the scheme presented here. This scheme exhibited almost perfect runway detections at a frame rate of 20 Hz. This frame rate is more than acceptable to be applied in an UAV. Similar to the Marker Detection and Tracking software, this runway detection scheme exhibits the capability to be deployed in this decade. Its use could also prove invaluable in the areas of cost and safety when it is applied to the patrol of national borders. With the current global outlook, the need for such a machine is certainly in the spotlight and the uses for said machine will only continue to grow.

In both algorithms and/or schemes there are some aspects that are not very desirable. There are serious issues regarding the performance in real-time that are of concern. At this time, it is not thought that these algorithms could be used in a real-time system. Although they execute fast enough to do so, the applicability to a real-time system is not very feasible. This is due to the fact that the execution times presented here are from simulations produced on a very fast ground-based computer. A computer of approximately one-third the power is feasible in a real-time system using current methodologies of incorporating computer systems into UAVs. Therefore, these algorithms will certainly need further evaluation to determine if they could ever be applied in a real-time system and possibly in a decade, these methods may be able to be used in a much faster real-time system but, with the equipment available today this proof of concept is simply that and no real-time applicability can be seen in the near future for these algorithms.

6.2 Recommendations

The future of the Marker Detection and Tracking software is certainly bright. Although current military interests are in the areas of passive markers, this software could easily be adapted to detect passive markers of any type with the correct hardware. The current recommendations for this software is to conduct further testing in the application of this software to pose estimation. The availability of a fully instrumented six degree of freedom robotic arm which could hold a simulated tanker aircraft would certainly be a step in the right direction. With this robotic arm, real measurements could be made as to the accuracy of this detection scheme. On a smaller level relating to the software itself, more robustness could be added to the software to make it more real-life friendly. Currently, the ability to lose sight of a marker will cause a fatal error but some additional software could be written to contend with this issue. In addition, the use of a color camera could be of some use in detecting passive markers, depending on the type.

The future idea the of research with the Runway Detection scheme is to be able to use the lines extracted from the image to define a trajectory for an aircraft to follow. This would be relatively easy since the sidelines of the runway already define the trajectory that is desired. All that is needed is to be able to pick out the lines. Of course, this is where the Runway Detection scheme comes in. Once the scheme has performed its job, the output is the end points, defined in pixels, of two lines representing the sides of the runway. Once this is complete, all that is necessary is to calculate a desired heading. Using the desired heading, a lateral-directional tracking controller could be developed that would minimize the error between the desired heading and the current heading. Once this is complete, some type of ground detection would be necessary to keep the UAV from hitting the ground. This ground detection would not be necessary if an altitude hold was employed using the global positioning system (GPS). Although, for the simple altitude hold to work reliably, the route to be followed would have to be mapped in order to calculate a reasonable altitude in which to fly such that a collision would not occur. This is a very feasible research idea that could be attempted with current equipment in the WVU UAV lab. This would make the Runway Detection research and its application to other things very attractive to many agencies in many countries.

References

1. "Machine Vision" Wikipedia, The Free Encyclopedia. 7 Jan 2006, 13:24. 12 Jan 2006, 03:28 http://en.wikipedia.org/w/index.php?title=Machine_vision&oldid=34237195.

2. "The History Of Computing" *Idea Finder*. 13 Jan 2006, 22:59 http://www.ideafinder.com/features/smallstep/computing.htm

3. McGarry, E. J. "An Outlook for Machine Vision" *Machine Vision Online*. 13 Jan 2006, 22:59 http://www.machinevisiononline.org/public/articles/articlesdetails.cfm?id=1134

4. Chazelle, Bernard. "The Computational Geometry Impact Task Force Report" Advances in Discrete and Computational Geometry 223 (1999): 407-463.

5. Deshmukh, K.S., Shinde, G.N. "An Adaptive Color Image Segmentation" Electronic Letters on Computer Vision and Image Analysis 5.4 (2005): 12-23.

6. Rezaei, Mehdi Home page. http://www.cs.tut.fi/%7Erezaei/Medical%20Image%20Segmentation.pdf

7. Ng, A., Jordan, M., Weiss, Y. "On spectral clustering: Analysis and an algorithm" Proceedings of the 2001 Advances in Neural Information Processing Systems 14 (2001).

8. Bach, F.R., Jordan, M.I. "Learning Spectral Clustering" Proceedings of the 2003 Advances in Neural Information Processing Systems 16 (2003).

9. Chen, A., Donovan, G, Sowmya, A, Trinder, J. "Inductive Clustering: automating low-level segmentation in high resolution images" Proceedings of the 2002 ICML, Machine Learning in Conputer Vision Workshop (2002).

10. Gibou, F. and Fedkiw, R., "A Fast Hybrid k-Means Level Set Algorithm for Segmentation" 4th Annual Hawaii International Conference on Statistics and Mathematics (2005): 281-291.

11. Eisenstein, J., Ghandeharizadeh, S., Huang, L., Shahabi, C., Shanbhag, G., Zimmermann, R. "Analysis of Clustering Techniques to Detect Hand Signs" Proceedings of the International Symposium on Intelligent Multimedia, Video and Speech Processing (2001).

12. Kuo, Chung-Feng, Shih, Chung-Yang, Lee, Jiunn-Yih "Repeat Pattern Segmentation of Printed Fabrics by Hough Transform Method" Textile Research Journal, Vol. 75, No. 11 (2005): 779-783.

13. Bab-Hadiashar, Alireza and Suter, David "Motion Segmentation Using Robust Statistics and Spatial Continuity" International Workshop on Image Analysis and Information Fusion IAIF '97 (1997)

14. Deselaers, Thomas, Keysers, Daniel, Ney, Hermann "Clustering visually similar images to improve image search engines" Diploma Thesis, *Chair of Computer Science VI*, RWTH Aachen University, 2003.

15. Yang, GZ Home page. "Edge Based Segmentation and Active Contours" http://www.doc.ic.ac.uk/~gzy/teaching/vision/vision-s03.pdf

16. Marek Brejl, Milan Sonka. "Automated Initialization and Automated Design of Border Detection Criteria in Edge-Based Image Segmentation" Proceedings of the 4th IEEE Southwest Symposium on Image Analysis and Interpretation (2000): 25.

17. Jiang, Ming Home page. "Edge Based Segmentation" http://iria.math.pku .edu.cn/~jiangm/courses/dip/html/node122.html

18. Pichumani, Ramani Home page. "Boundary-Based Segmentation" http:// homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAMANI1/node 24.html#SECTION00315000000000000000

19. Yang, GZ Home page. "Hough Transform" http://www.doc.ic.ac.uk/~gzy /teaching/vision/vision-s03.pdf

20. Jiang, Ming Home page. "Hough Transform" http://iria.math.pku.edu.cn /~jiangm/courses/dip/html/node132.html

21. Ray, Sid Home page. "Image Segmentation" http://www.csse.monash. edu.au/~sid/teach/CSE3314/notes10.pdf

22. Yang, GZ Home page. "Region Based Segmentation" http://www.doc.ic. ac.uk/~gzy/teaching/vision/vision-s04.pdf

23. Kothe, Ullrich "Primary Image Segmentation" DAGM-Symposium (1995): 554-561.

24. McClain, T. W. "Coordinated Control of Unmanned Air Vehicles" Air Vehicles Directorate, Wright-Patterson Air Force Base, Ohio, Summer 1999.

25. Tandale, Monish D., Bowers, Roshawn, and Valasek, John, “Robust Trajectory Tracking Controller for Vision Based Probe and Drogue Autonomous Aerial Refueling”, AIAA-2005-5868, Proceedings of the AIAA Guidance, Navigation, and Control Conference, San Francisco, CA, 15-18 August 2005.

26. Valasek, John, Kimmett, Jennifer, Junkins, John L. “Autonomous Aerial Refueling Utilizing A Vision Based Navigation System”, Journal of Guidance, Control, and Dynamics, Volume 28, Number 5, pp. 979-989, September-October 2005.

27. Campa, G., Mammarella, M., Napolitano, M.R., Fravolini, M.L., Pollini, L. "Addressing Pose Estimation Issues for Machine Vision based UAV Autonomous Aerial Refueling", Submitted for publication to IEEE Transaction On Systems, Man and Cybernetics, Accepted May 2005.

28. Dell’Aquila, R.V., Campa, G., Napolitano, M.R., Mammarella, M. “Real-Time Machine-Vision-Based Position Sensing System for UAV Aerial Refueling”, Submitted to the SPRINGER, Journal of Real-Time Image Processing, May 2006.

29. Campa, G., Napolitano, M.R., Vendra, Soujanya, Fravolini, M.L. “A Simulation Environment for Machine Vision based Aerial Refueling for UAVs”, Submitted to the IEEE Transaction On Aerospace and Electronic Systems, April 2006.

30. Fravolini, M.L., Campa, G., Napolitano, M.R., Ficola, A. “Evaluation of Machine Vision Algorithms for Autonomous Aerial Refueling for Unmanned Aerial Vehicles”, Submitted to the AIAA Journal of Aerospace Computing, Information and Communication, April 2005.

31. Vendra, S., Campa, G., Napolitano, M.R., Mammarella, M., Fravolini, M.L. “Addressing Corner Detection Issues for Machine Vision based UAV Aerial Refueling”, Submitted to the Journal of Machine Vision Application, October 2005.

32. Nalepka, Joseph P., Hinchman, Jacob L. “Aerial Refueling for Unmanned Air Vehicles”, Air Force Research Laboratory.

33. Pollini, L., Mati, R., Innocenti, M., Campa, G., Napolitano, M.R. “A Synthetic Environment for Simulation of Vision-Based Formation Flight” Proceedings of the AIAA Modeling and Simulation Technologies Conference, Austin, TX, August 2003.

34. Seanor, B., Campa, G., Gu, Y., Napolitano, M.R., Rowe, L., Perhinschi, M. “Formation Flight Test Results for UAV Research Aircraft Models”, Proceedings of the 2004 AIAA Intelligent Systems Technology Conference, Chicago, IL, September 2004.

35. Sattigeri, Ramachandra, Calise, Anthony J. “AnAdaptive Approach to Vision-Based Formation Control”, Proceedings of the 2003 AIAA Guidance, Navigation, and Control Conference and Exhibit, Austin TX, August 2003.

36. Proctor, Allison, Johnson, Eric N. “Vision-Only Aircraft Flight Control Methods and Test Results”, Proceedings of the 2004 AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, RI, August 2004.

37. Frew, Eric, et al. “Vision-Based Road-Following Using a Small Autonomous Aircraft” Proceedings of the 2004 IEEE Aerospace Conference (2004).

38. Frew, Eric, Langelaan, Jack, Joo, Sungmoon “Adaptive Receding Horizon Control for Vision-Based Navigation of Small Unmanned Aircraft”, To be presented at the 2006 American Control Conference, Minneapolis, MN, June 2006.

39. Frew, Eric “Comparison of Lateral Controllers for Following Linear Structures Using Computer Vision”, To be presented at the 2006 American Control Conference, Minneapolis, MN, June 2006.

40. Cornall, T. “A Low Computation Method to Determine Horizon Angle from Video” Monash University, Department of Electrical and Computer Systems, Technical Report MECSE-4-2004.

41. Ettinger, S., Mechyba, M.C., Ifju, P.G., Wasnak, M “Vision-Guided Flight Stability and Control for Micro Air Vehicles” Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2002).

42. Green, W., Sevcik, K.,Oh, P., “A Competition to Identify Key Challenges for Unmanned Aerial Robots in Near-Earth Environments” Proceedings of the IEEE International Conference on Advanced Robotics (2005).

43. He, Zhihai, Iyer, Ram Venkataraman, Chandler, Phillip R. “Vision-Based UAV Flight Control and Obstacle Avoidance”, To be presented at the 2006 American Control Conference, Minneapolis, MN, June 2006.

44. Star SAFIRE^{TM} HD" *FLIR Systems.* http://www.flir.com/imaging/Airborne/Products/StarSAFIREHD.aspx

45. Kamath, C., Sengupta, S., Poland, D., Futterman, J.. "Use of Machine Vision Techniques to Detect Human Settlements in Satellite Images" Proceedings of the International Society of Optical Engineering 5014 (2003): 270-280.

46. Weixing Wang. "An Edge Based Segmentation Algorithm for Rock Fracture Tracing" Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (2005): 43-48.

47. Gachter, S. "Results on Range Image Segmentation for Service Robots" Ecole Polytechnique Federale de Lausanne, Laboratoire de Systemes Autonomes, Technical Report EPFL-LSA-2005-01.

48. Muniz, R., Rivera, F. "Segmentation of Hyperspectral Images Uning the Hough Transform" Poster. http://www.censsis.neu.edu/Education/StudentResearch/2001/posters/rivera_f!.pdf

49. Sippel, M., Traxler, M. "Packagers Choose Machine Vision Quality Inspection to Reduce Waste and Boost ROI" *Automation.com.* 2004. 13 Jan 2006, 23:01 http://www.automation.com/sitepages/pid1632.php

50. Toh, Teck Soon "Multiple Target Tracking Using Computer-Aided Vision", Masters Thesis, West Virginia University, May 1991.

51. "Machine Vision and Industrial Inspection" Fairchild Imaging. http://www.fairchildimaging.com/main/machinevision.htm

52. Ramos, V., Muge, F. "Image Colour Segmentation by Genetic Algorithms" Proceedings of the 11th Portuguese Conference on Pattern Recognition (2000).

53. Ferreira, S., Garin, V., Gosselin, B. "A Text Detection Technique Applied in the Framework of a Mobile Camera-Based Application" Proceedings of Camera-based Document Analysis and Recognition (2005): 133-139.

54. Everingham, M.R., Thomas, B.T., Troscianko, T., Easty, D. "Neural-network virtual reality mobility air for the severely visually impaired" Proceedings of the 2nd European Conference on Disability, Virtual Reality and Associated Technologies (1998): 183-192.

55. Godbout, Andrew "Segmentation Methods Applicable to Segmenting The Moving Organ in a CT Scan" Saint Mary's University 2003.

56. Dow, Mark "An Edge Based Segmentation Method" International Society of Magnetic Resonance in Medicine (2004): Poster.

57. Texmol, A., Sari-Sarraf, H., Mirta, S., Long, R., Gururajan, A. "Customized Hough Transform for Robust Segmentation of Cervical Vertebrae from X-Ray Images" Fifth IEEE Southwest Symposium on Image Analysis and Interpretation (2002).

58. Geng, W., et al. "Quantitative classification and natural clustering of C. elegans behavioral patterns" Genetics 165 (2003): 1117-1123.

59. Steward, B.L. and Tian, L.F. "Real-time machine vision weed sensing" ASAE Paper No 98-7006. St. Joseph, MI: ASAE.

60. "Video and Image Processing Blockset" The Mathworks http://www.mathworks.com/access/helpdesk/help/toolbox/vipblks/

61. "Gamma correction" Wikipedia, The Free Encyclopedia. 15 May 2006, 17:48 UTC. 18 May 2006, 05:08 http://en.wikipedia.org/w/index.php?title=Gamma_correction&oldid=53351099>.

62. Basu, Saurav Home page "Gamma Correction" .http://www.cs.utah.edu/~sbasu/ipprojects /project1 /index.html

63. "Image Acquisiton Toolbox" The Mathworks http://www.mathworks.com/access/helpdesk/help/toolbox/imaq/

64. Class Notes EE 465 - Intro to Image Processing. West Virginia University. Professor Dr. Xin Li Spring 2004.

65. Fisher, R., Perkins, S., Walker, A., Wolfart, E. "Sobel Edge Detection" Hypermedia Image Processing Reference http://homepages.inf.ed.ac.uk /rbf/HIPR2/sobel.htm

66. "Standard Simulink Blockset" The Mathworks http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/

Appendix A

MATLAB Code

For

Matlab Based Modified K-Means Clustering Algorithm

marker_detect.m

```
% marker_detect.m
% Machine Vision Image Processing
% Larry Rowe
% Fall 2004

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%%%%%  This version of software performs NO ESTIMATION of the position
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

clear all;
clc;
close all;
imaqreset;

video = aviread('..\VIDEOS\MV1.avi');

level=0.7; % Threshold Level
T=15;  %Pixel filter threshold level
index = 0;

for i=1:150
    index=index+1;
    tic;  % Begin counting frame time;

    % Get single frame to work with
    frame1=frame2im(video(i));

    % Convert to binary and threshold
    frame2=im2bw(frame1,level);

    clear targetindex;

    % FIND TARGET PIXELS
    i=1;j=1;k=0;
    for i=1:640
        for j=1:480
            if frame2(j,i)==1
                k=k+1;
                targetindex(k,:)=[i j];
            end
        end
    end

    % TARGET DETERMINATION AND LOCATION CALCULATION
    clear targetindex1;
    targetindex1=targetindex;

    % FIND LIST OF TARGET 1 PIXELS
    k=0;q=0;
    X1=targetindex1(1,1);
    Y1=targetindex1(1,2);
    Xrange_max1=X1+T;
    Yrange_min1=Y1-T;
    Yrange_max1=Y1+T;
    listsize1=size(targetindex1);
    clear target1;
    for k = 1:listsize1(1)
        if targetindex1(k,1) <= Xrange_max1 && targetindex1(k,1) ~= 0
            if targetindex1(k,2)>= Yrange_min1 && targetindex1(k,2)<=
Yrange_max1
                q=q+1;
                target1(q,1:2) =targetindex1(k,:);
                targetindex1(k,:)=0;
            end
        end
    end

    varsize1=size(target1);
    clear targetindex2;
    j=0;
    for k = 1:listsize1(1)
        if targetindex1(k,1) ~= 0
            j=j+1;
            targetindex2(j,1:2)=targetindex1(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 1
    target1sumX=0;target1sumY=0;
    for h=1:varsize1(1)
        target1sumX=target1(h,1)+target1sumX;
        target1sumY=target1(h,2)+target1sumY;
    end
    target1avgX=target1sumX/varsize1(1);
    target1avgY=target1sumY/varsize1(1);

    % FIND LIST OF TARGET 2 PIXELS
    k=0;q=0;
    X2=targetindex2(1,1);
    Y2=targetindex2(1,2);
    Xrange_max2=X2+T;
    Yrange_min2=Y2-T;
    Yrange_max2=Y2+T;
    listsize2=size(targetindex2);
    clear target2;
    for k = 1:listsize2(1)
        if targetindex2(k,1) <= Xrange_max2 && targetindex2(k,1) ~= 0
            if targetindex2(k,2)>= Yrange_min2 && targetindex2(k,2)<=
Yrange_max2
                q=q+1;
                target2(q,1:2)=targetindex2(k,:);
                targetindex2(k,:)=0;
            end
        end
    end

    varsize2=size(target2);
    clear targetindex3;
    j=0;
    for k = 1:listsize2(1)
        if targetindex2(k,1) ~= 0
```

```
            j=j+1;
            targetindex3(j,1:2)=targetindex2(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 2
    target2sumX=0;target2sumY=0;
    for h=1:varsize2(1)
        target2sumX=target2(h,1)+target2sumX;
        target2sumY=target2(h,2)+target2sumY;
    end
    target2avgX=target2sumX/varsize2(1);
    target2avgY=target2sumY/varsize2(1);

    % FIND LIST OF TARGET 3 PIXELS
    k=0;q=0;
    X3=targetindex3(1,1);
    Y3=targetindex3(1,2);
    Xrange_max3=X3+T;
    Yrange_min3=Y3-T;
    Yrange_max3=Y3+T;
    listsize3=size(targetindex3);
    clear target3;
    for k = 1:listsize3(1)
        if targetindex3(k,1) <= Xrange_max3 && targetindex3(k,1) ~= 0
            if targetindex3(k,2)>= Yrange_min3 && targetindex3(k,2)<=
Yrange_max3
                q=q+1;
                target3(q,1:2)=targetindex3(k,:);
                targetindex3(k,:)=0;
            end
        end
    end

    varsize3=size(target3);
    clear targetindex4;
    j=0;
    for k = 1:listsize3(1)
        if targetindex3(k,1) ~= 0
            j=j+1;
            targetindex4(j,1:2)=targetindex3(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 3
    target3sumX=0;target3sumY=0;
    for h=1:varsize3(1)
        target3sumX=target3(h,1)+target3sumX;
        target3sumY=target3(h,2)+target3sumY;
    end
    target3avgX=target3sumX/varsize3(1);
    target3avgY=target3sumY/varsize3(1);

    % FIND LIST OF TARGET 4 PIXELS
    k=0;q=0;
    X4=targetindex4(1,1);
    Y4=targetindex4(1,2);
    Xrange_max4=X4+T;
    Yrange_min4=Y4-T;
    Yrange_max4=Y4+T;
    listsize4=size(targetindex4);
    clear target4;
    for k=1:listsize4(1)
        if targetindex4(k,1) <= Xrange_max4 && targetindex4(k,1) ~= 0
            if targetindex4(k,2)>= Yrange_min4 && targetindex4(k,2)<=
Yrange_max4
                q=q+1;
                target4(q,1:2)=targetindex4(k,:);
                targetindex4(k,:)=0;
            end
        end
    end

    varsize4=size(target4);
    clear targetindex5;
    j=0;
    for k = 1:listsize4(1)
        if targetindex4(k,1) ~= 0
            j=j+1;
            targetindex5(j,1:2)=targetindex4(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 4
    target4sumX=0;target4sumY=0;
    for h=1:varsize4(1)
        target4sumX=target4(h,1)+target4sumX;
        target4sumY=target4(h,2)+target4sumY;
    end
    target4avgX=target4sumX/varsize4(1);
    target4avgY=target4sumY/varsize4(1);

    % FIND LIST OF TARGET 5 PIXELS
    k=0;q=0;
    X5=targetindex5(1,1);
    Y5=targetindex5(1,2);
    Xrange_max5=X5+T;
    Yrange_min5=Y5-T;
    Yrange_max5=Y5+T;
    listsize5=size(targetindex5);
    clear target5;
    for k= 1:listsize5(1)
        if targetindex5(k,1) <= Xrange_max5 && targetindex5(k,1) ~= 0
            if targetindex5(k,2)>= Yrange_min5 && targetindex5(k,2)<=
Yrange_max5
                q=q+1;
                target5(q,1:2)=targetindex5(k,:);
                targetindex5(k,:)=0;
            end
        end
    end

    varsize5=size(target5);
    % FIND AVERAGE PIXEL LOCATION OF TARGET 5
    target5sumX=0;target5sumY=0;
    for h=1:varsize5(1)
        target5sumX=target5(h,1)+target5sumX;
        target5sumY=target5(h,2)+target5sumY;
    end
    target5avgX=target5sumX/varsize5(1);
    target5avgY=target5sumY/varsize5(1);

    % DEFINE TARGET LOCATIONS FROM AVERAGE CALCULATIONS
    Ftarget1=[target1avgX target1avgY];
    Ftarget2=[target2avgX target2avgY];
    Ftarget3=[target3avgX target3avgY];
    Ftarget4=[target4avgX target4avgY];
    Ftarget5=[target5avgX target5avgY];

    %  DETERMINE ABSOLUTE DISTANCES
    dist(1,:)=[sqrt(((Ftarget1(1,1)-Ftarget2(1,1))^2)+((Ftarget1(1,2)-
Ftarget2(1,2))^2)),1,2];
    dist(2,:)=[sqrt(((Ftarget1(1,1)-Ftarget3(1,1))^2)+((Ftarget1(1,2)-
Ftarget3(1,2))^2)),1,3];
    dist(3,:)=[sqrt(((Ftarget1(1,1)-Ftarget4(1,1))^2)+((Ftarget1(1,2)-
Ftarget4(1,2))^2)),1,4];
    dist(4,:)=[sqrt(((Ftarget1(1,1)-Ftarget5(1,1))^2)+((Ftarget1(1,2)-
Ftarget5(1,2))^2)),1,5];
    dist(5,:)=[sqrt(((Ftarget2(1,1)-Ftarget3(1,1))^2)+((Ftarget2(1,2)-
Ftarget3(1,2))^2)),2,3];
    dist(6,:)=[sqrt(((Ftarget2(1,1)-Ftarget4(1,1))^2)+((Ftarget2(1,2)-
Ftarget4(1,2))^2)),2,4];
    dist(7,:)=[sqrt(((Ftarget2(1,1)-Ftarget5(1,1))^2)+((Ftarget2(1,2)-
Ftarget5(1,2))^2)),2,5];
    dist(8,:)=[sqrt(((Ftarget3(1,1)-Ftarget4(1,1))^2)+((Ftarget3(1,2)-
Ftarget4(1,2))^2)),3,4];
    dist(9,:)=[sqrt(((Ftarget3(1,1)-Ftarget5(1,1))^2)+((Ftarget3(1,2)-
Ftarget5(1,2))^2)),3,5];
    dist(10,:)=[sqrt(((Ftarget4(1,1)-Ftarget5(1,1))^2)+((Ftarget4(1,2)-
Ftarget5(1,2))^2)),4,5];

    %  DETECTING THE WING TIPS
    wings=max(dist(:,1));
    for i=1:10
        if dist(i,1)==wings;
            wingdef(1,1)=dist(i,2);
            wingdef(1,2)=dist(i,3);
        end
    end
    leftwing=wingdef(1);
    rightwing=wingdef(2);

    %  DETECTING THE HORIZONTAL STAB TIPS
    count=0;
    for i=1:10
        if dist(i,2)~=wingdef(1)&&dist(i,2)~=wingdef(2)&&dist(i,3)...
                ~=wingdef(1)&&dist(i,3)~=wingdef(2)
            count=count+1;
            elev(count,:)=dist(i,:);
        end
    end
    stabsize=size(elev);
    limit=stabsize(1);
    stab=max(elev(:,1));
    for i=1:limit
        if elev(i,1)==stab;
            stabdef(1,1)=elev(i,2);
            stabdef(1,2)=elev(i,3);
        end
    end
    leftstab=stabdef(1);
    rightstab=stabdef(2);

    %  DEFINING THE VERTICAL STAB TIP
    vertstab=15-leftwing-rightwing-leftstab-rightstab;

    % SAVE TARGET LOCATIONS FOR COMPARISONS
    targetloc=[Ftarget1;Ftarget2;Ftarget3;Ftarget4;Ftarget5];
    targetlocX=targetloc(:,1);
    targetlocY=targetloc(:,2);

FtargetX=[Ftarget1(1);Ftarget2(1);Ftarget3(1);Ftarget4(1);Ftarget5(1)];

FtargetY=[Ftarget1(2);Ftarget2(2);Ftarget3(2);Ftarget4(2);Ftarget5(2)];
    ACTtargetlocationX(:,index)=FtargetX;
    ACTtargetlocationY(:,index)=FtargetY;

    %CALCULATE BANK ANGLE
    riserun=(targetlocY(rightwing)-targetlocY(leftwing))/...
        (targetlocX(rightwing)-targetlocX(leftwing));
    phirad=atan(riserun);
    phideg(index)=atan(riserun)*180/pi();

    frametime(index,:)=toc;
end;

% PLOTTING ROUTINE
figure;
plot(frametime);
axis([1 150 0 1])
title('Image Processing Speed on Frame by Frame Basis')
xlabel('Frame Number');
ylabel('Time between frames (secs)');

figure;
plot(phideg);
axis([1 150 -90 90])
title('Aircraft Bank Angle As Calculated From Wing Tip Target
Positions')
xlabel('Frame Number');
ylabel('Bank Angle - Phi (degrees)');

figure;
axis ij;
hold on;
plot(ACTtargetlocationX(5,:),ACTtargetlocationY(5,:),'b');
plot(ACTtargetlocationX(4,:),ACTtargetlocationY(4,:),'b');
plot(ACTtargetlocationX(3,:),ACTtargetlocationY(3,:),'b');
plot(ACTtargetlocationX(2,:),ACTtargetlocationY(2,:),'b');
plot(ACTtargetlocationX(1,:),ACTtargetlocationY(1,:),'b');
title('Actual Location for All Markers');
xlabel('X-Coordinate');
ylabel('Y-Coordinate');
axis([0 640 0 480])
hold off;

% SAVE DATA FILE FOR COMPARISON
save data.mat -MAT ACTtargetlocation* frametime phideg;
%  END
```

Appendix B

MATLAB Code

For

Matlab Based Modified K-Means
Clustering Algorithm with Loss of Marker Visibility

marker_loss.m

```
% marker_loss.m
% Machine Vision Image Processing
% Larry Rowe
% May 2006

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
%%%%%  This version of software performs marker detection ONLY.  No
%%%%%
%%%%%  labeling of the markers is performed in this software. This
%%%%%
%%%%%  software does accomodate the loss/gain of any number of markers.
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

clear all;
clc;
close all;
imaqreset;

video = aviread('..\VIDEOS\MV5.avi');

level=0.7; % Threshold Level
T=15;  %Pixel filter threshold level
index = 0;
disp('READY TO BEGIN SIMULATION, PRESS A KEY TO CONTINUE');
pause;


for i=1:150
    index=index+1;

    tic;

    % Get single frame to work with
    frame1= frame2im(video(i));

    % Convert to binary and threshold
    frame2=im2bw(frame1,level);

    clear targetindex;

    %  FIND TARGET PIXELS
    i=1;j=1;k=0;
    for i=1:640
        for j=1:480
            if frame2(j,i)==1
                k=k+1;
                targetindex(k,:)=[i j];
            end
        end
    end

    %  TARGET DETERMINATION AND LOCATION CALCULATION
    k=0;q=0;targetsize=0;targetnum=0;targetlist=0;
    g=size(targetindex);
    listsize=g(1);
    while(sum(targetindex(:,1)) ~= 0 && sum(targetindex(:,2)) ~= 0)

        targetnum=targetnum+1;
        k=targetsize+1;

        Xrange_max=targetindex(k,1)+T;
        Yrange_min=targetindex(k,2)-T;
        Yrange_max=targetindex(k,2)+T;


        for u = k:listsize
            if targetindex(u,1) <= Xrange_max && targetindex(u,1) ~= 0
                if targetindex(u,2) >= Yrange_min && targetindex(u,2)
<= Yrange_max
                    q=q+1;
                    targetlist(q,1:3) = [targetindex(u,:),targetnum];
                    targetindex(u,:) = 0;
                end
            end
        end
        h=size(targetlist);
        targetsize=h(1);
    end

    targetnum=0;targetsumX=0;targetsumY=0;
    for i=1:targetsize
        if targetlist(i,3) == (targetnum+1)
            targetnum=targetnum+1;
            targetsumX=0;targetsumY=0;count=0;
        end
        if targetlist(i,3) == targetnum
            targetsumX=targetlist(i,1)+targetsumX;
            targetsumY=targetlist(i,2)+targetsumY;
            count=count+1;
        end
        targetavgX=targetsumX/count;
        targetavgY=targetsumY/count;
        centroidlist(targetnum,1:2)=[targetavgX targetavgY];
    end


    targetlocX=centroidlist(:,1);
    targetlocY=centroidlist(:,2);
    clear centroidlist;


    if length(targetlocX) == 5
        targetlocX(6)=0;
        targetlocY(6)=0;
    end

    ACTtargetlocationX(index,1:6)=targetlocX;
    ACTtargetlocationY(index,1:6)=targetlocY;

    % PLOT TARGETS ON ORIGINAL FRAME
    imshow(frame1);
    hold on;
    plot(targetlocX,targetlocY,'rO');
    frametime(index,:)=toc;
    pause(.03);
end;

% PLOTTING ROUTINE
figure;
plot(frametime);
axis([1 150 0 1])
title('Image Processing Speed on Frame by Frame Basis')
xlabel('Frame Number');
ylabel('Time between frames (secs)');

% SAVE DATA FILE FOR COMPARISON
save data.mat -MAT ACTtargetlocation* frametime;
%  END
```

Appendix C

MATLAB Code

For

Matlab Based Advanced K-Means Clustering and Tracking Algorithm

marker_track.m

```
% marker_track.m
% Machine Vision Image Processing
% Larry Rowe
% Fall 2004

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%%%%  This version of software performs the full estimation of the
%%%%%
%%%%%  of the position using velocity and acceleration or velocity
%%%%%
%%%%%  only.  To use VELOCITY ONLY for estimation, UNCOMMENT LINE
%%%%%
%%%%%  366 and 367.      To use VELOCITY AND ACCELERATION for
%%%%%
%%%%%  estimation, UNCOMMENT LINE 362 and 363.
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

clear all;
clc;
close all;
imaqreset;

video = aviread('..\VIDEOS\MV1.avi');

level=0.7; % Threshold Level
T=15;  %Pixel filter threshold level
index = 0;

for i=1:3
    index=index+1;

    tic;

    % Get single frame to work with
    frame1= frame2im(video(i));

    % Convert to binary and threshold
    frame2=im2bw(frame1,level);

    % Perform binary erosion to make targets smaller
    %se=strel('square',3); %Structuring Element
    %frame3=imerode(frame2,se);
    frame3=frame2;

    clear targetindex;

    % FIND TARGET PIXELS
    i=1;j=1;k=0;
    for i=1:640
        for j=1:480
            if frame3(j,i)==1
                k=k+1;
                targetindex(k,:)=[i j];
            end
        end
    end

    % TARGET DETERMINATION AND LOCATION CALCULATION
    clear targetindex1;
    targetindex1=targetindex;

    % FIND LIST OF TARGET 1 PIXELS
    k=0;q=0;
    X1=targetindex1(1,1);
    Y1=targetindex1(1,2);
    Xrange_max1=X1+T;
    Yrange_min1=Y1-T;
    Yrange_max1=Y1+T;
    listsize1=size(targetindex1);
    clear target1;
    for k = 1:listsize1(1)
        if targetindex1(k,1) <= Xrange_max1 && targetindex1(k,1) ~= 0
            if targetindex1(k,2)>= Yrange_min1 && targetindex1(k,2)<=
Yrange_max1
                q=q+1;
                target1(q,1:2) =targetindex1(k,:);
                targetindex1(k,:)=0;
            end
        end
    end

    varsize1=size(target1);
    clear targetindex2;
    j=0;
    for k = 1:listsize1(1)
        if targetindex1(k,1) ~= 0
            j=j+1;
            targetindex2(j,1:2)=targetindex1(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 1
    target1sumX=0;target1sumY=0;
    for h=1:varsize1(1)
        target1sumX=target1(h,1)+target1sumX;
        target1sumY=target1(h,2)+target1sumY;
    end
    target1avgX=target1sumX/varsize1(1);
    target1avgY=target1sumY/varsize1(1);

    % FIND LIST OF TARGET 2 PIXELS
    k=0;q=0;
    X2=targetindex2(1,1);
    Y2=targetindex2(1,2);
    Xrange_max2=X2+T;
    Yrange_min2=Y2-T;
    Yrange_max2=Y2+T;
    listsize2=size(targetindex2);
    clear target2;
    for k = 1:listsize2(1)
        if targetindex2(k,1) <= Xrange_max2 && targetindex2(k,1) ~= 0
```

```
            if targetindex2(k,2)>= Yrange_min2 && targetindex2(k,2)<=
Yrange_max2
                q=q+1;
                target2(q,1:2)=targetindex2(k,:);
                targetindex2(k,:)=0;
            end
        end
    end

    varsize2=size(target2);
    clear targetindex3;
    j=0;
    for k = 1:listsize2(1)
        if targetindex2(k,1) ~= 0
            j=j+1;
            targetindex3(j,1:2)=targetindex2(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 2
    target2sumX=0;target2sumY=0;
    for h=1:varsize2(1)
        target2sumX=target2(h,1)+target2sumX;
        target2sumY=target2(h,2)+target2sumY;
    end
    target2avgX=target2sumX/varsize2(1);
    target2avgY=target2sumY/varsize2(1);

    % FIND LIST OF TARGET 3 PIXELS
    k=0;q=0;
    X3=targetindex3(1,1);
    Y3=targetindex3(1,2);
    Xrange_max3=X3+T;
    Yrange_min3=Y3-T;
    Yrange_max3=Y3+T;
    listsize3=size(targetindex3);
    clear target3;
    for k = 1:listsize3(1)
        if targetindex3(k,1) <= Xrange_max3 && targetindex3(k,1) ~= 0
            if targetindex3(k,2)>= Yrange_min3 && targetindex3(k,2)<=
Yrange_max3
                q=q+1;
                target3(q,1:2)=targetindex3(k,:);
                targetindex3(k,:)=0;
            end
        end
    end

    varsize3=size(target3);
    clear targetindex4;
    j=0;
    for k = 1:listsize3(1)
        if targetindex3(k,1) ~= 0
            j=j+1;
            targetindex4(j,1:2)=targetindex3(k,:);
        end
    end
    % FIND AVERAGE PIXEL LOCATION OF TARGET 3
    target3sumX=0;target3sumY=0;
    for h=1:varsize3(1)
        target3sumX=target3(h,1)+target3sumX;
        target3sumY=target3(h,2)+target3sumY;
    end
    target3avgX=target3sumX/varsize3(1);
    target3avgY=target3sumY/varsize3(1);

    % FIND LIST OF TARGET 4 PIXELS
    k=0;q=0;
    X4=targetindex4(1,1);
    Y4=targetindex4(1,2);
    Xrange_max4=X4+T;
    Yrange_min4=Y4-T;
    Yrange_max4=Y4+T;
    listsize4=size(targetindex4);
    clear target4;
    for k=1:listsize4(1)
        if targetindex4(k,1) <= Xrange_max4 && targetindex4(k,1) ~= 0
            if targetindex4(k,2)>= Yrange_min4 && targetindex4(k,2)<=
Yrange_max4
                q=q+1;
                target4(q,1:2)=targetindex4(k,:);
                targetindex4(k,:)=0;
            end
        end
    end

    varsize4=size(target4);
    clear targetindex5;
    j=0;
    for k = 1:listsize4(1)
        if targetindex4(k,1) ~= 0
            j=j+1;
            targetindex5(j,1:2)=targetindex4(k,:);
        end
    end

    % FIND AVERAGE PIXEL LOCATION OF TARGET 4
    target4sumX=0;target4sumY=0;
    for h=1:varsize4(1)
        target4sumX=target4(h,1)+target4sumX;
        target4sumY=target4(h,2)+target4sumY;
    end
    target4avgX=target4sumX/varsize4(1);
    target4avgY=target4sumY/varsize4(1);

    % FIND LIST OF TARGET 5 PIXELS
    k=0;q=0;
    X5=targetindex5(1,1);
    Y5=targetindex5(1,2);
    Xrange_max5=X5+T;
    Yrange_min5=Y5-T;
    Yrange_max5=Y5+T;
    listsize5=size(targetindex5);
    clear target5;
    for k= 1:listsize5(1)
        if targetindex5(k,1) <= Xrange_max5 && targetindex5(k,1) ~= 0
            if targetindex5(k,2)>= Yrange_min5 && targetindex5(k,2)<=
Yrange_max5
                q=q+1;
                target5(q,1:2)=targetindex5(k,:);
                targetindex5(k,:)=0;
            end
        end
    end

    varsize5=size(target5);

    % FIND AVERAGE PIXEL LOCATION OF TARGET 5
    target5sumX=0;target5sumY=0;
    for h=1:varsize5(1)
        target5sumX=target5(h,1)+target5sumX;
        target5sumY=target5(h,2)+target5sumY;
    end
    target5avgX=target5sumX/varsize5(1);
    target5avgY=target5sumY/varsize5(1);

    % DEFINE TARGET LOCATIONS FROM AVERAGE CALCULATIONS
    Ftarget1=[target1avgX target1avgY];
    Ftarget2=[target2avgX target2avgY];
    Ftarget3=[target3avgX target3avgY];
    Ftarget4=[target4avgX target4avgY];
    Ftarget5=[target5avgX target5avgY];

    %  DETERMINE ABSOLUTE DISTANCES
    dist(1,:)=[sqrt(((Ftarget1(1,1)-Ftarget2(1,1))^2)+((Ftarget1(1,2)-
Ftarget2(1,2))^2)),1,2];
    dist(2,:)=[sqrt(((Ftarget1(1,1)-Ftarget3(1,1))^2)+((Ftarget1(1,2)-
Ftarget3(1,2))^2)),1,3];
    dist(3,:)=[sqrt(((Ftarget1(1,1)-Ftarget4(1,1))^2)+((Ftarget1(1,2)-
Ftarget4(1,2))^2)),1,4];
    dist(4,:)=[sqrt(((Ftarget1(1,1)-Ftarget5(1,1))^2)+((Ftarget1(1,2)-
Ftarget5(1,2))^2)),1,5];
    dist(5,:)=[sqrt(((Ftarget2(1,1)-Ftarget3(1,1))^2)+((Ftarget2(1,2)-
Ftarget3(1,2))^2)),2,3];
    dist(6,:)=[sqrt(((Ftarget2(1,1)-Ftarget4(1,1))^2)+((Ftarget2(1,2)-
Ftarget4(1,2))^2)),2,4];
    dist(7,:)=[sqrt(((Ftarget2(1,1)-Ftarget5(1,1))^2)+((Ftarget2(1,2)-
Ftarget5(1,2))^2)),2,5];
    dist(8,:)=[sqrt(((Ftarget3(1,1)-Ftarget4(1,1))^2)+((Ftarget3(1,2)-
Ftarget4(1,2))^2)),3,4];
    dist(9,:)=[sqrt(((Ftarget3(1,1)-Ftarget5(1,1))^2)+((Ftarget3(1,2)-
Ftarget5(1,2))^2)),3,5];
    dist(10,:)=[sqrt(((Ftarget4(1,1)-Ftarget5(1,1))^2)+((Ftarget4(1,2)-
Ftarget5(1,2))^2)),4,5];

    %  DETECTING THE WING TIPS
    wings=max(dist(:,1));
    for i=1:10
        if dist(i,1)==wings;
            wingdef(1,1)=dist(i,2);
            wingdef(1,2)=dist(i,3);
        end
    end
    leftwing=wingdef(1);
    rightwing=wingdef(2);

    %  DETECTING THE HORIZONTAL STAB TIPS
    count=0;
    for i=1:10
        if
dist(i,2)~=wingdef(1)&&dist(i,2)~=wingdef(2)&&dist(i,3)~=wingdef(1)&&di
st(i,3)~=wingdef(2)
            count=count+1;
            elev(count,:)=dist(i,:);
        end
    end
    stabsize=size(elev);
    limit=stabsize(1);
    stab=max(elev(:,1));
    for i=1:limit
        if elev(i,1)==stab;
            stabdef(1,1)=elev(i,2);
            stabdef(1,2)=elev(i,3);
        end
    end
    leftstab=stabdef(1);
    rightstab=stabdef(2);

    %  DEFINING THE VERTICAL STAB TIP
    vertstab=15-leftwing-rightwing-leftstab-rightstab;

    % SAVE TARGET LOCATIONS FOR INERTIAL CALCULATIONS
    targetloc=[Ftarget1;Ftarget2;Ftarget3;Ftarget4;Ftarget5];
    targinert(1:5,1:2,index)=targetloc;
    targetlocX=targetloc(:,1);
    targetlocY=targetloc(:,2);

    % PLOT TARGETS ON ORIGINAL FRAME
%     imshow(frame1);
%     hold on;
%     plot(targetlocX(leftwing),targetlocY(leftwing),'gp');
%     plot(targetlocX(rightwing),targetlocY(rightwing),'rp');
%     plot(targetlocX(leftstab),targetlocY(leftstab),'go');
%     plot(targetlocX(rightstab),targetlocY(rightstab),'ro');
%     plot(targetlocX(vertstab),targetlocY(vertstab),'bd');
%     hold off;

    %CALCULATE BANK ANGLE
    riserun=(targetlocY(rightwing)-
targetlocY(leftwing))/(targetlocX(rightwing)-targetlocX(leftwing));
    phirad=atan(riserun);
    phideg(index)=atan(riserun)*180/pi();

    frametime(index,:)=toc;
end;

% END OF FIRST LOOP FINDING THE INITIAL TARGETS
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% CALCULATE DELTA T
dT12=frametime(1);
dT23=frametime(2);
dT13=frametime(1)+frametime(2);

% CALCULATE VELOCITY AND ACCELERATION OF EACH TARGET
for i=1:5
    targetvelocX12(i,:)=(targinert(i,1,2)-targinert(i,1,1))/dT12;
    targetvelocY12(i,:)=(targinert(i,2,2)-targinert(i,2,1))/dT12;

    targetvelocX23(i,:)=(targinert(i,1,3)-targinert(i,1,2))/dT23;
    targetvelocY23(i,:)=(targinert(i,2,3)-targinert(i,2,2))/dT23;

    targetaccelX(i,:)=(targetvelocX12(i)-targetvelocX23(i))/dT13;
    targetaccelY(i,:)=(targetvelocY12(i)-targetvelocY23(i))/dT13;
    % OUTPUT IS PIXELS/FRAME VELOCITY
end

% CALCULATE ACCELERATION OF EACH TARGET

%  NEW THRESHOLD FOR REGION OF INTEREST
RoIT=10;

% CREATE INITIAL ESTIMATE TARGET LOCATIONS
for i=1:3
    ESTtargetlocationX(:,i)=targinert(:,1,i);
    ESTtargetlocationY(:,i)=targinert(:,2,i);
    ACTtargetlocationX(:,i)=targinert(:,1,i);
    ACTtargetlocationY(:,i)=targinert(:,2,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% LOOP FOR ESTIMATING WINDOW AND SO ON

for framecount=4:150
    index=index+1;
    clear ACTtargetloc;

    %%  USE FOR FULL ESTIMATION  %%

ESTtargetlocX=targetlocX+targetvelocX23*dT23+0.5*targetaccelX*dT23^2;

ESTtargetlocY=targetlocY+targetvelocY23*dT23+0.5*targetaccelY*dT23^2;

    %%  USE FOR ESTIMATION WITH NO ACCELERATION  %%
    %ESTtargetlocX=targetlocX+targetvelocX23*dT23;
    %ESTtargetlocY=targetlocY+targetvelocY23*dT23;
    ESTtargetlocationX(:,index)=ESTtargetlocX;  %USED FOR PLOTTING
    ESTtargetlocationY(:,index)=ESTtargetlocY;  %USED FOR PLOTTING

    tic;

    % Get single frame to work with
    frame1=frame2im(video(framecount));

    % Convert to binary and threshold
    frame2=im2bw(frame1,level);

    % Perform binary erosion to make targets smaller
    frame3=frame2;

    % FIND LIST OF PIXELS
    ESTtargetlocrangeXdec=[ESTtargetlocX-RoIT ESTtargetlocX+RoIT];
    ESTtargetlocrangeYdec=[ESTtargetlocY-RoIT ESTtargetlocY+RoIT];

    ESTtargetlocrangeX=uint16(ESTtargetlocrangeXdec);
    ESTtargetlocrangeY=uint16(ESTtargetlocrangeYdec);

    % FIND TARGET PIXELS
    for targ=1:5
        k=0;
        for i=ESTtargetlocrangeX(targ,1):ESTtargetlocrangeX(targ,2)
            for j=ESTtargetlocrangeY(targ,1):ESTtargetlocrangeY(targ,2)
                if frame3(j,i)==1
                    k=k+1;
                    ACTtargetloc(k,:,targ)=[i j];
                end
            end
        end
    end

    listsize=size(ACTtargetloc);

    % FILTER OUT ZEROS AND CALCULATE THE CENTROIDS
    num=0;
    for targ=1:5
        j=0;
        clear centroidindex;
        targetsumX=0;targetsumY=0;
        for k = 1:listsize(1)
            if ACTtargetloc(k,1,targ) ~= 0
                j=j+1;
                centroidindex(j,:)=[ACTtargetloc(k,1,targ)
ACTtargetloc(k,2,targ)];
            end
        end
        CIsize=size(centroidindex);
        num=num+1;
        targetsumX=sum(centroidindex(:,1));
        targetsumY=sum(centroidindex(:,2));
        targetavgX(:,num)=targetsumX/CIsize(1);
        targetavgY(:,num)=targetsumY/CIsize(1);
    end
    % DEFINE TARGET LOCATIONS FROM AVERAGE CALCULATIONS
    Ftarget1=[targetavgX(1) targetavgY(1)];
    Ftarget2=[targetavgX(2) targetavgY(2)];
    Ftarget3=[targetavgX(3) targetavgY(3)];
    Ftarget4=[targetavgX(4) targetavgY(4)];
    Ftarget5=[targetavgX(5) targetavgY(5)];

FtargetX=[Ftarget1(1);Ftarget2(1);Ftarget3(1);Ftarget4(1);Ftarget5(1)];

FtargetY=[Ftarget1(2);Ftarget2(2);Ftarget3(2);Ftarget4(2);Ftarget5(2)];
    ACTtargetlocationX(:,index)=FtargetX;
    ACTtargetlocationY(:,index)=FtargetY;

    % SAVE TARGET LOCATIONS FOR INERTIAL CALCULATIONS
    targetloc=[Ftarget1;Ftarget2;Ftarget3;Ftarget4;Ftarget5];
    targinert(1:5,1:2,index)=targetloc;
    targetlocX=targetloc(:,1);
    targetlocY=targetloc(:,2);

    % PLOT TARGETS ON ORIGINAL FRAME
%     imshow(frame1);
%     hold on;
%     plot(targetlocX(leftwing),targetlocY(leftwing),'gp');
%     plot(targetlocX(rightwing),targetlocY(rightwing),'rp');
%     plot(targetlocX(leftstab),targetlocY(leftstab),'go');
%     plot(targetlocX(rightstab),targetlocY(rightstab),'ro');
%     plot(targetlocX(vertstab),targetlocY(vertstab),'bd');
%     hold off;
    %pause;

    %CALCULATE BANK ANGLE
    riserun=(targetlocY(rightwing)-
targetlocY(leftwing))/(targetlocX(rightwing)-targetlocX(leftwing));
    phirad=atan(riserun);
    phideg(index)=atan(riserun)*180/pi();

    frametime(index,:)=toc;

    % CALCULATE DELTA T
    dT12=frametime(index-2);
    dT23=frametime(index-1);
    dT13=dT12+dT23;

    % CALCULATE VELOCITY AND ACCELERATION OF EACH TARGET TO ESTIMATE
NEW
    for i=1:5
        targetvelocX12(i,:)=(targinert(i,1,index-1)-
targinert(i,1,index-2))/dT12;
        targetvelocY12(i,:)=(targinert(i,2,index-1)-
targinert(i,2,index-2))/dT12;

        targetvelocX23(i,:)=(targinert(i,1,index)-targinert(i,1,index-
1))/dT23;
        targetvelocY23(i,:)=(targinert(i,2,index)-targinert(i,2,index-
1))/dT23;

        targetaccelX(i,:)=(targetvelocX12(i)-targetvelocX23(i))/dT13;
        targetaccelY(i,:)=(targetvelocY12(i)-targetvelocY23(i))/dT13;
        % OUTPUT IS PIXELS/SEC
    end

end;
% END OF LAST LOOP FINDING THE ESTIMATED TARGETS

% PLOTTING
figure;
subplot(2,1,1);
hold on;
plot(frametime);
axis([1 10 0 1])
title('Image Processing Speed on Frame by Frame Basis - Transient
Illustration')
xlabel('Frame Number');
ylabel('Time between frames (secs)');
hold off;
subplot(2,1,2);
hold on;
plot(frametime);
axis([1 150 0 0.1]);
title('Image Processing Speed on Frame by Frame Basis')
xlabel('Frame Number');
ylabel('Time between frames (secs)');
hold off;

figure;
plot(phideg);
axis([1 150 -20 30])
title('Aircraft Roll Angle As Calculated From Wing Tip Marker
Positions')
xlabel('Frame Number');
ylabel('Bank Angle - {\Phi} (degrees)');

figure;
subplot(2,1,1),plot(ESTtargetlocationX(1,:),'r');
hold on;
subplot(2,1,1),plot(ACTtargetlocationX(1,:),'b');
title('Estimate vs. Actual X-Coordinate Location for Target 1');
xlabel('Frame Number');
ylabel('X-Coordinate');
subplot(2,1,2),plot(ESTtargetlocationY(1,:),'r');
hold on;
subplot(2,1,2),plot(ACTtargetlocationY(1,:),'b');
title('Estimate vs. Actual Y-Coordinate Location for Target 1');
xlabel('Frame Number');
ylabel('Y-Coordinate');

figure;
subplot(2,1,1),plot(ESTtargetlocationX(2,:),'r');
hold on;
subplot(2,1,1),plot(ACTtargetlocationX(2,:),'b');
title('Estimate vs. Actual X-Coordinate Location for Target 2');
xlabel('Frame Number');
ylabel('X-Coordinate');
subplot(2,1,2),plot(ESTtargetlocationY(2,:),'r');
```

```
hold on;
subplot(2,1,2),plot(ACTtargetlocationY(2,:),'b');
title('Estimate vs. Actual Y-Coordinate Location for Target 2');
xlabel('Frame Number');
ylabel('Y-Coordinate');

figure;
subplot(2,1,1),plot(ESTtargetlocationX(3,:),'r');
hold on;
subplot(2,1,1),plot(ACTtargetlocationX(3,:),'b');
title('Estimate vs. Actual X-Coordinate Location for Target 3');
xlabel('Frame Number');
ylabel('X-Coordinate');
subplot(2,1,2),plot(ESTtargetlocationY(3,:),'r');
hold on;
subplot(2,1,2),plot(ACTtargetlocationY(3,:),'b');
title('Estimate vs. Actual Y-Coordinate Location for Target 3');
xlabel('Frame Number');
ylabel('Y-Coordinate');

figure;
subplot(2,1,1),plot(ESTtargetlocationX(4,:),'r');
hold on;
subplot(2,1,1),plot(ACTtargetlocationX(4,:),'b');
title('Estimate vs. Actual X-Coordinate Location for Target 4');
xlabel('Frame Number');
ylabel('X-Coordinate');
subplot(2,1,2),plot(ESTtargetlocationY(4,:),'r');
hold on;
subplot(2,1,2),plot(ACTtargetlocationY(4,:),'b');
title('Estimate vs. Actual Y-Coordinate Location for Target 4');
xlabel('Frame Number');
ylabel('Y-Coordinate');

figure;
subplot(2,1,1),plot(ESTtargetlocationX(5,:),'r');
hold on;
subplot(2,1,1),plot(ACTtargetlocationX(5,:),'b');
title('Estimate vs. Actual X-Coordinate Location for Target 5');
xlabel('Frame Number');
ylabel('X-Coordinate');
subplot(2,1,2),plot(ESTtargetlocationY(5,:),'r');
hold on;
subplot(2,1,2),plot(ACTtargetlocationY(5,:),'b');
title('Estimate vs. Actual Y-Coordinate Location for Target 5');
xlabel('Frame Number');
ylabel('Y-Coordinate');

figure;
axis ij;
hold on;
plot(ESTtargetlocationX(5,:),ESTtargetlocationY(5,:),'r');
plot(ACTtargetlocationX(5,:),ACTtargetlocationY(5,:),'b');
plot(ESTtargetlocationX(4,:),ESTtargetlocationY(4,:),'r');
plot(ACTtargetlocationX(4,:),ACTtargetlocationY(4,:),'b');
plot(ESTtargetlocationX(3,:),ESTtargetlocationY(3,:),'r');
plot(ACTtargetlocationX(3,:),ACTtargetlocationY(3,:),'b');
plot(ESTtargetlocationX(2,:),ESTtargetlocationY(2,:),'r');
plot(ACTtargetlocationX(2,:),ACTtargetlocationY(2,:),'b');
plot(ESTtargetlocationX(1,:),ESTtargetlocationY(1,:),'r');
plot(ACTtargetlocationX(1,:),ACTtargetlocationY(1,:),'b');
title('Estimate vs. Actual Location for All Markers');
xlabel('X-Coordinate');
ylabel('Y-Coordinate');
axis([0 640 0 480])
hold off;

%  COMPUTE ERROR
error5X=abs(ESTtargetlocationX(5,:)-ACTtargetlocationX(5,:));
error5Y=abs(ESTtargetlocationY(5,:)-ACTtargetlocationY(5,:));
error5=sqrt(error5X.*error5X+error5Y.*error5Y);

error4X=abs(ESTtargetlocationX(4,:)-ACTtargetlocationX(4,:));
error4Y=abs(ESTtargetlocationY(4,:)-ACTtargetlocationY(4,:));
error4=sqrt(error4X.*error4X+error4Y.*error4Y);

error3X=abs(ESTtargetlocationX(3,:)-ACTtargetlocationX(3,:));
error3Y=abs(ESTtargetlocationY(3,:)-ACTtargetlocationY(3,:));
error3=sqrt(error3X.*error3X+error3Y.*error3Y);

error2X=abs(ESTtargetlocationX(2,:)-ACTtargetlocationX(2,:));
error2Y=abs(ESTtargetlocationY(2,:)-ACTtargetlocationY(2,:));
error2=sqrt(error2X.*error2X+error2Y.*error2Y);

error1X=abs(ESTtargetlocationX(1,:)-ACTtargetlocationX(1,:));
error1Y=abs(ESTtargetlocationY(1,:)-ACTtargetlocationY(1,:));
error1=sqrt(error1X.*error1X+error1Y.*error1Y);

avgerr5=mean(error5);
avgerr4=mean(error4);
avgerr3=mean(error3);
avgerr2=mean(error2);
avgerr1=mean(error1);
avgerr=[avgerr1;avgerr2;avgerr3;avgerr4;avgerr5];

stderr5=std(error5);
stderr4=std(error4);
stderr3=std(error3);
stderr2=std(error2);
stderr1=std(error1);
stderr=[stderr1;stderr2;stderr3;stderr4;stderr5];

%  PLOT MEAN OF THE ERROR
figure;
hold on;
bar(avgerr);
set(gca,'XTick',0:1:6)
set(gca,'XTickLabel',{'','Left Wing','Left Stab','Rudder','Right
Stab','Right Wing'})
title('Mean of Position Error for All Markers');
ylabel('Pixels');
hold off;

%  PLOT STANDARD DEVIATION OF THE ERROR
```

```
figure;
hold on
bar(stderr);
set(gca,'XTick',0:1:6)
set(gca,'XTickLabel',{'','Left Wing','Left Stab','Rudder','Right
Stab','Right Wing'})
title('Standard Deviation of Position Error for All Markers');
ylabel('Pixels');
hold off;

% CREATE DATA FILE VECTORS
ACTtargetlocationX_FULL=ACTtargetlocationX;
ACTtargetlocationY_FULL=ACTtargetlocationY;
ESTtargetlocationX_FULL=ESTtargetlocationX;
ESTtargetlocationY_FULL=ESTtargetlocationY;
frametime_FULL=frametime;
phideg_FULL=phideg;
avgerr_FULL=avgerr;
stderr_FULL=stderr;

% SAVE DATA FILE FOR COMPARISON
save data.mat -MAT ACTtargetlocationX_FULL ACTtargetlocationY_FULL
ESTtargetlocationX_FULL...
    ESTtargetlocationY_FULL frametime_FULL phideg_FULL avgerr_FULL
stderr_FULL;
%  END
```