Addressing corner detection issues for machine vision based UAV aerial refueling

Soujanya Vendra
West Virginia University

Addressing Corner Detection Issues for Machine Vision based UAV Aerial Refueling

Soujanya Vendra

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, West Virginia
2006

Keywords: machine vision, aerial refueling, feature extraction, corner detection

ABSTRACT

Addressing Corner Detection Issues for Machine Vision based UAV Aerial Refueling

Soujanya Vendra

The need for developing autonomous aerial refueling capabilities for an Unmanned Aerial Vehicle (UAV) has risen out of the growing importance of UAVs in military and non-military applications. The AAR capabilities would improve the range and the loiter time capabilities of UAVs. A number of AAR techniques have been proposed, based on GPS based measurements and Machine Vision based measurements. The GPS based measurements suffer from distorted data in the wake of the tanker. The MV based techniques proposed the use of optical markers which-when detected-were used to determine relative orientation and position of the tanker and the UAV. The drawback of the MV based techniques is the assumption that all the optical markers are always visible and functional. This research effort proposes an alternative approach where the pose estimation does not depend on optical markers but on Feature Extraction methods. The thesis describes the results of the analysis of specific 'corner detection' algorithms within a Machine Vision - based approach for the problem of Aerial Refueling for Unmanned Aerial Vehicles. Specifically, the performances of the SUSAN and the Harris corner detection algorithms have been compared. Special emphasis was placed on evaluating their accuracy, the required computational effort, and the robustness of both methods to different sources of noise. Closed loop simulations were performed using a detailed Simulink®-based simulation environment to reproduce docking maneuvers, using the US Air Force refueling boom.

To Joshi and Rajani Satti

Acknowledgments

Many people contributed to the successful completion of this thesis, most notably Dr. Marcello Napolitano, who has been an exceptional advisor and mentor through out my graduate career. I have greatly benefited from his help and guidance throughout the project. I am grateful for his constant support and encouragement throughout my Master's Program.

I would like to acknowledge and specially thank my committee members Dr. Giampiero Campa and Dr. Arun Ross for taking time from their busy schedules to review and contribute their thoughts to this research effort. Their help has been the vital point for the successful completion of the project.

I would also like to thank members of the AAR research group, Marco Mammarella and Rocco Dell'Aquila for their extended support.

Finally I am most grateful to my family and friends, whose love and support made everything possible.

Table Of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Aerial Refueling

Aerial refueling is referred to as the practice of transferring fuel from one aircraft to another during flight, allowing the receiving aircraft to remain airborne longer, and/or to take off with a greater payload.

Some of the earliest experiments in aerial refueling took place in the 1902's. The simplest form of air refueling technique has two slow-flying aircraft flying in formation, with a hose run down from a handheld gas tank on one airplane and placed into the usual fuel filler of the other. In 1949 from February 26 to March 3 an American B-50 Superfortress "Lucky Lady II" flew around the world in 94 hours without stopping. Refueling was performed 3 times during the flight from 4 pairs of KB-29M tankers. The flight started and ended at Fort Worth Texas. Refueling was performed over the skies of West Africa, Guam, and in the Pacific between Hawaii and the US West Coast [1]. This first non-stop circumnavigation of the globe proved that aerial refueling extends the aircraft's range, thereby allowing airpower forces to increase levels of mass, economy of force, flexibility, versatility, and maneuverability. Figure 1.1 shows one of the earliest aerial refueling techniques.

1.2 Aerial Refueling Systems

The two most common aerial refueling approaches are the "*boom and receptacle*" system and the "*probe and drogue*" system. A much less popular approach is the "*wing-to-wing*" method, which is no longer used. The US Air Force uses the "*boom and receptacle*" system while, the US Navy, Marine Corps, Air Force Helicopters, and other NATO nations use the "*probe and drogue*" system.

Figure 1.1: An early 1960's aerial refueling technique

1.2.1 Boom and Receptacle System

The “*boom and receptacle*” system used by the US Air Force is based on the refueling boom [2]. The *boom* is a long, rigid, hollow shaft, fitted to the rear of the aircraft. It has a telescopic extension, called the boom nozzle to keep fuel in and permit it to flow, and small “V” shaped wings (as shown in Figure 1.2) to enable it to be “flown” into the receptacle of the receiver aircraft, to be refueled. This “*receptacle*” is fitted onto the top of the aircraft – usually on its centerline – and usually either behind or close in front of the cockpit. The “*receptacle*” is a round opening which connects to the fuel tanks, with a valve to keep the fuel in when not being refueled, and dust and debris out. The boom has a nozzle that fits into this opening [1].

During refueling operations, a tanker aircraft will fly in a straight and level altitude at constant speed, while the receiver takes a standard position behind and below the tanker. Once in position, the receiver pilot flies formation with the tanker, although this can be complicated by wake turbulence. The “*boom-operator*” then unlatches the boom from its stowed position, and directs it toward the receiver by “flying” it with the attached wings. Figure 1.3 shows the boom operation in action. The telescopic section is then hydraulically extended until the nozzle fits into the receiver’s receptacle. When an electrical signal is passed between the boom and receiver, both valves are hydraulically opened, and the pumps operated by the tanker pilot, provide fuel through the shaft of the boom into the receiver. Once the two aircrafts are in refueling position, additional lights (pilot director lights (PDI's)) on the tanker will be turned on if the receiver flies too far or too near, too low or too high. These lights are activated by sensing switches in the boom.

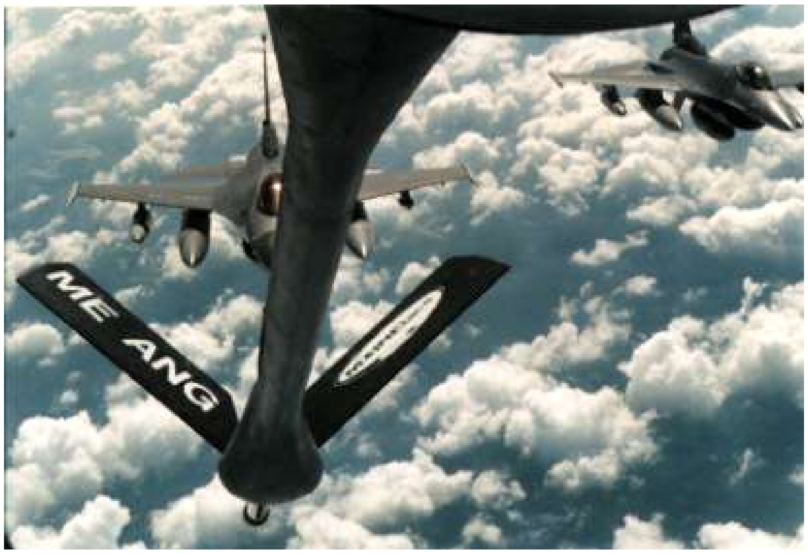


Figure 1.2: The "V" shaped wings of the boom [64]

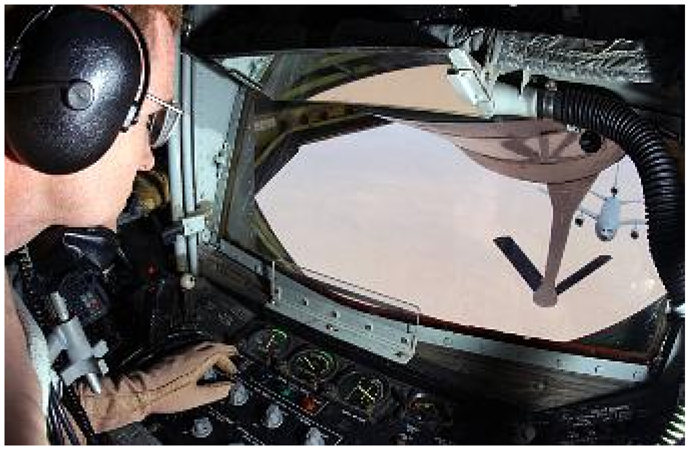


Figure 1.3: Boom operator [63]

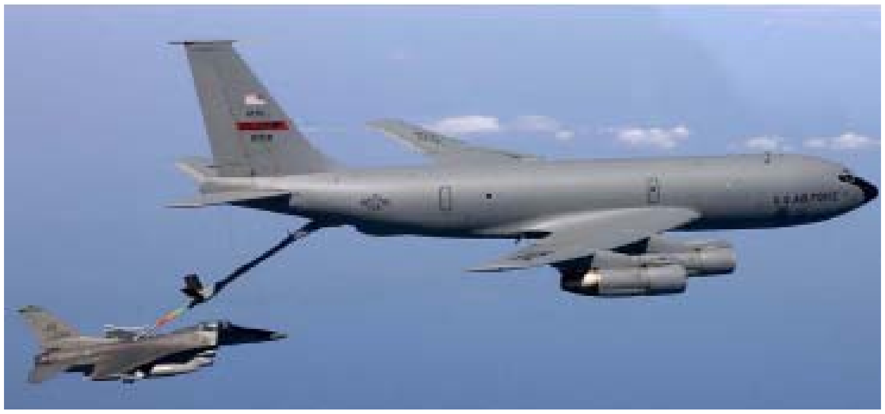


Figure 1.4: KC-135 tanker refuels an F-16 Fighting Falcon using the boom system [63]

When the refueling is complete, the valves are closed and the boom is automatically or manually retracted by the boom operator. In addition to the US Air Force, the "*boom and receptacle*" system is used by the Netherlands (KDC-10), Israel (modified Boeing 707) and Turkey (ex-USAF KC-135R). All the mentioned nations operate US designed aircraft [1].

The primary advantage to this method of refueling is that higher volumes of fuel can be transferred in a shorter amount of time. Although tankers equipped with rigid refueling booms can only service one properly equipped aircraft at a time, the transfer capacity is useful for the US Air Force, which operates many very large aircraft such as strategic bombers.

1.2.2 Probe and Drogue System

The US Navy, Marine Corps as well as the armed forces of other North Atlantic Treaty Organizations (NATO) nations use this approach [2, 3]. The "*drogue*" is a fitting resembling a plastic shuttlecock, attached to a flexible hose at its narrow end with a valve. The receiver has a "*probe*" arm placed usually on the side of the airplane's nose, as shown in the Figure 1.5.

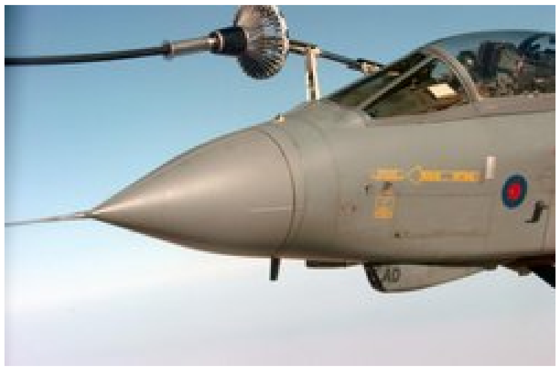


Figure 1.5: Tornado GR4 with probe attached to the drogue of a tanker [1]

The tanker flies straight and in level, and the drogue is allowed to trail behind and below it. It is primarily the receiver aircraft pilot's responsibility to achieve a contact between the "*probe*" and "*drogue*". Once the "*probe*" is in the "*drogue*", the aerodynamic drag acting on the "*drogue*", forces the "*probe*" and "*drogue*" to connect together. After a successful contact between the "*probe*" and "*drogue*" the tanker refuels the receiver aircraft. When refueling is complete, the receiver aircraft decelerates hard enough to yank the probe out of the "*drogue*". A "*probe and drogue*" refueling system is shown in Figure 1.6.

Some boom-carrying tankers can be modified to refuel "probe" equipped aircraft with the help of a "boom-drogue adapter" (BDA). The BDA consists of special hose which is attached to the telescopic end of the boom, and which terminates in hard non-collapsible "drogue". The BDA can only be fitted or removed on ground [4]. The BDA approach is shown in Figure 1.7. Other tankers may have both a boom and one or more hose-and-drogue assemblies attached to the wing tips known as the Multi-Point Refueling System (MPRS) as shown in Figure 1.8. Unlike the "boom and receptacle" system, multiple aircraft can be refueled simultaneously with the "probe and drogue" system.

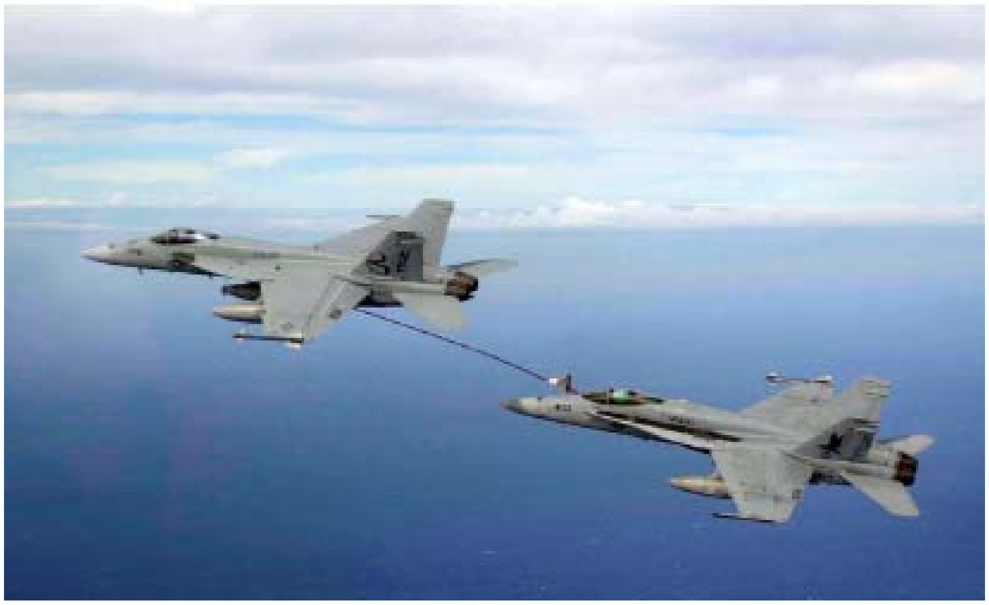


Figure 1.6: F/A-18E Super Hornet performs an in flight refueling evolution with an F/A-18C Hornet using the probe and drogue technique [65]

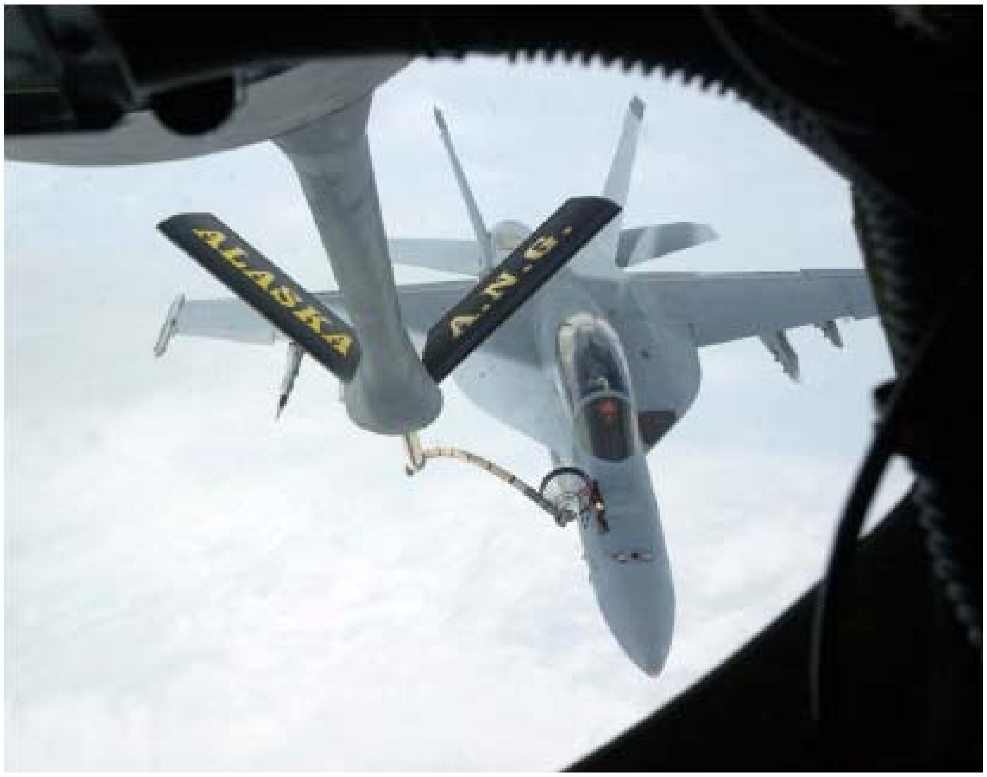


Figure 1.7: Navy F/A-18F Super Hornet is refueled by a KC-135R Stratotanker using a boom-drogue adapter [63]

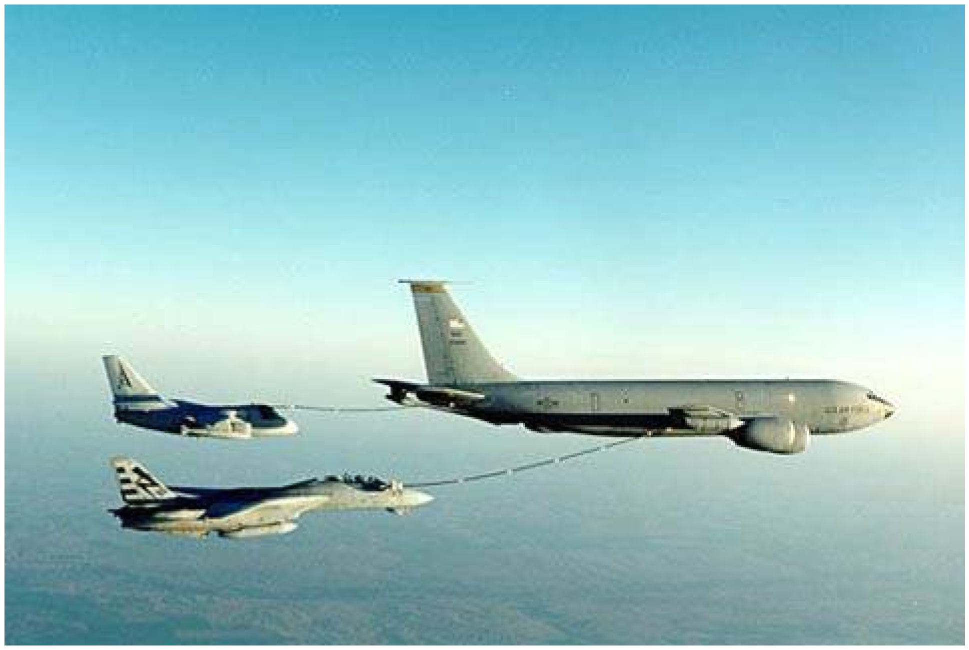


Figure 1.8: A multi point refueling system [66]

1.2.3 Wing to Wing Method

In the Wing-to-Wing method, the tanker aircraft releases a flexible hose from its wingtip which is caught by an aircraft flying beside it. After the hose is locked by the receiver aircraft, which is equipped with a lock under the wingtip for this purpose, and a connection established, the fuel will be pumped. Though the wing-to-wing method was used previously on a small number of Soviet Tu-4 and Tu-16, it is no longer in use today [1].

1.3 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs) -also referred as RPVs (Remotely Piloted Vehicle), drones, robot planes, and pilot less aircraft- are defined by the Department of Defense (DoD) as powered aerial vehicles that do not carry a human operator. These vehicles use aerodynamic forces to produce lift, can fly autonomously or are piloted remotely, can be expendable or recoverable, and can carry a lethal or non-lethal payload [5]. DoD originally sought UAVs primarily to satisfy surveillance requirements in Close Range, Short Range, or Endurance categories. Close Range is defined to be within a distance of

50 kilometers, Short Range within 200 kilometers and Endurance as anything beyond. DoD currently possesses five major UAVs: the US Air Force’s Predator and Global Hawk (Figure 1.9), the US Navy and Marine Corps’s Pioneer, and the US Army’s Hunter and Shadow [5] (Figure 1.10).

The importance of the UAVs has tremendously grown in recent years both in the military and non-military applications. The evolution of technologies that allow safe, reliable UAV flights over populated areas, led to the increase of their non-military applications. Some of the emerging non-military applications of the UAVs are the use of less sophisticated UAVs as aerial camera platforms for movie making and entertainment business, in television news reporting and coverage arenas, homeland security, and medical re-supply.

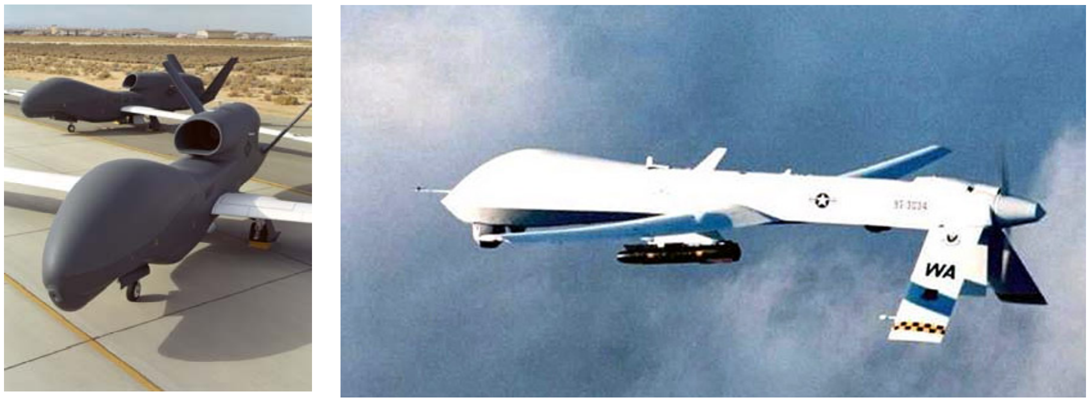


Figure 1.9: U.S. Air Force Global Hawk [67], U.S. Air Force Predator [63]

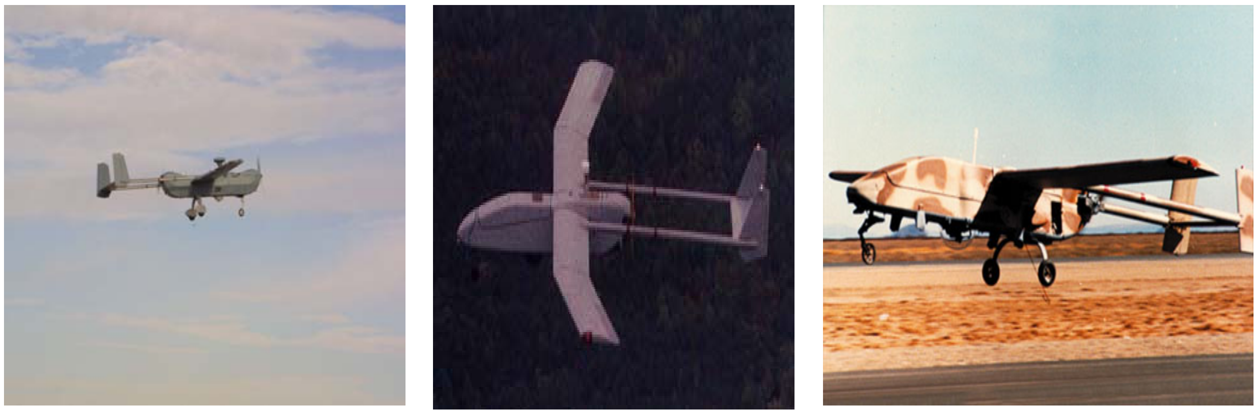


Figure 1.10: U.S. Army Hunter [67], U.S. Army Shadow [68], and the U.S. Navy Pioneer [69].

Reducing costs and risks of human casualties is one immediate advantage of using UAVs for military purposes. Additional advantages are the possibility of avoiding troop deployment in enemy territory for dangerous missions as is done currently with manned missions, and the possibility of long endurance reconnaissance missions. It is envisioned that formations of UAVs will perform not only intelligence and reconnaissance missions but also provide close air support, precision strike, and suppression of enemy air defenses [6, 7].

One of the biggest limitations of deployed military UAVs is their limited aircraft range. In order to perform long-duration missions the UAVs have to be enabled to loiter over the theater of operation for extended periods of time with extended aircraft range. By deploying UAVs to forward bases, the UAVs would be closer to the theater of operation, thus decreasing their en route time and increasing loiter time. But several factors like terrain and weather that determines how close the UAVs can be deployed to the targets, and most importantly the safety of the ground deployment troops play against this forward basing, [8]. Hence, an approach for long duration missions, avoiding the risk of ground deployment troop casualties, is the critical goal of acquiring AAR (Autonomous Aerial Refueling) capabilities for the UAVs [8].

1.4 Research Objective

Several methods have been proposed for autonomous aerial refueling for both the "boom and receptacle" and the "probe and drogue" systems. These methods include Global Positioning Systems (GPS), Machine Vision (MV) Techniques with active markers or vision sensors, and a combination of the sensors.

Differential Global Positioning Systems (DGPS) [9,10] uses reference ground stations to provide a correction to signals from GPS satellites. The estimation is generally more accurate than GPS alone, with a typical position error of 1-3 meters [11]. The DGPS can operate at a great distance, which is definitely an advantage over MV applications. Disadvantages of the DGPS include problems with multipath errors caused by interference of signal that has reached the receiver antenna by two or more paths. Other problems include the satellite drop out, geometric dilution of precision, and cycle slip [12, 13, 14]. At close proximity range the tanker frame itself could distort the GPS signal. Hence, it can be finalized that the accuracy for proximity navigation is beyond the capabilities of the DGPS. For these reasons, new approaches were proposed [14,15,16], which present a fuzzy sensor fusion strategy of MV and GPS technologies.

MV systems work by processing 2D images from a single or multiple cameras. A mapping is used to determine 3D information from 2D images. This involves relating markers, such as optical markers, Light Emitting Diodes (LEDs) or beacons, in an image to their known position on the tanker. Certain vision systems do not require the cooperation of the target in anyway and is basically based on identifying key points in the 2D images.

Pollini et al proposed a MV technique capable of estimating the drogue position using a set of infrared light-emitting diodes mounted on the drogue. [17,18]. The LEDs are mounted in a co-planar configuration, at the vertices of a regular polygon. Infra Red (IR) camera mounted on the UAV, captures the images of the drogue and passes these images to a modified version of the estimation algorithm created by Lu, Hager and Mjolsness (LHM). The LHM algorithm determines the relative position and attitude based on minimizing the collinearity error. The estimation algorithm is shown to converge within ten iterations. The approach assumes that all the IR light emitters are identical and that the light emission is not modulated. Having *n* uniquely identifiable markers is necessary for the application of the LHM algorithm. Problems can arise with the performance of the algorithm if the LEDs fail due to hardware failure and/or physical interference between the UAV on-board camera and the markers [19].

Fravolini et al proposed similar MV technique using optical markers. The optical markers were installed at ad-hoc points on the tanker, and image-processing techniques were provided by the MV algorithm, to isolate the red optical markers from an image stream generated by the 3D virtual world. [15,19]. The problem is given in the form of correspondences each composed of 3D reference points of the markers expressed in object coordinates and its 2D projection expressed in the image coordinates. Gaussian Least Square Differential Correction (GLSDC) algorithm has been implemented in this study, for the pose estimation. The pose estimation is based on the minimization of a non-linear cost function typically solved using the Gauss Newton method. This MV approach also assumes that all the optical markers are fully functional at all times during the docking phase. But as the UAV approaches the tanker some of the markers might exit the visual range of the on-board camera. Additionally certain physical interferences like the boom itself and/or structural components of the tanker may obstruct certain markers [20].

Valasek et al proposed an approach for vision sensing and vision based proximity navigation of spacecraft [21]. A new sensor -which utilizes area Position Sensing Diode photo detectors in the focal plane of an omni directional camera- was invented for this purpose. Target lights called beacons, which are an array of light emitting diodes (LEDs) are fixed in the target spacecraft at known positions, and an optical sensor is attached rigidly to the chase spacecraft. The beacons are activated through the sensor computer, turning them on alternately, and angles toward their line of sight are measured every time the sensor detects them. This approach assumes a wireless infrared or radio datalink between the tanker and the receiver aircraft. The disadvantage of this approach is that it is sensitive to interception and jamming during hostile conditions [12].

The major disadvantage of these methods is the simple assumption that all the LEDs, optical markers or beacons are fully functional throughout the docking sequence. Hardware failures, physical interferences, and loss of the datalink can cause serious problems in the detection of the markers and estimation of the pose. The pose estimation algorithm assumes that *n* number of optical markers are always available, and can compute the pose if and only if these *n* number of markers are available. These problems with the optical markers can be overcome by an alternative concept, wherein the pose estimation does not depend on the optical marker but on the feature extraction techniques. Feature extraction techniques can be implemented to obtain information regarding the physical features of the tanker/target. Since physical features are always available within the tanker frame, the problem of loosing some of the features does not arise. This approach does not assume any cooperation from the tanker; thus circumvents the interception problems. The physical corners of the aircraft, or corners formed by the components on the aircraft, or templates of certain aircraft parts could be considered as the features to be detected. Hence, the MV algorithm comprises of feature extraction techniques to extract the features of the tanker/target, matching of the 3D features with their 2D projections, and the estimation of the pose.

The objective of this work is to evaluate the performance of specific feature extraction algorithms- corner detection techniques - within a general MV based AR approach. Particularly, in this context the MV system has to detect and correctly identify features (specifically corners) on the tanker airframe. This information is then used to evaluate the relative distance and orientation between the tanker and the UAV aircraft, assuming that the position of the detected features in the tanker reference frame is constant and is known.

This study has been performed using an AR simulation environment, developed at WVU in Simulink and interfaced with Virtual Reality Toolbox (VRT). This closed-loop simulation interacts with a Virtual Reality (VR) environment by moving visual 3D models of the aircraft in a virtual world and by acquiring a stream of images from the environment. A "feature extraction" algorithm uses these images for the detection of corners resulting from specific features of the tanker aircraft. Specifically, both the Harris [40] and SUSAN [31] methods have been evaluated as "Corner Detection" (CD) algorithms. The detected corners are then matched with a set of physical features on the tanker through the use of a Detection and Labeling (DAL) algorithm. Finally, the positions of the matched corners are used to evaluate the position and the orientation of the UAV with respect to the tanker by a Pose Estimation (PE) algorithm. The general block diagram of the MV – based scheme is shown in Figure 1.11. The simulation developed in this effort includes feature extraction algorithms along with detection and labeling and pose estimation algorithms. The above MV schemes are applied to a 'Tanker + UAV' scheme which includes the modeling of atmospheric turbulence [22], wake effects [23,24], as well as the docking control laws [16].

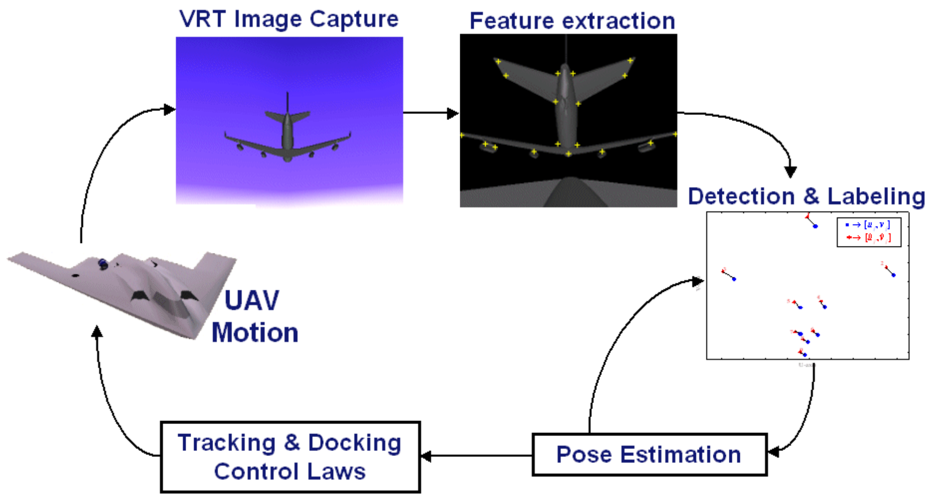


Figure 1.11: Information flow involved in the MV based AR problem

1.5 Organization of the Thesis

Chapter 1 introduces the aerial refueling systems currently in use and the MV techniques, which were developed to acquire AAR capabilities for the UAV. The drawbacks of these MV techniques were mentioned along with the alternative approach to overcome these drawbacks, specifically feature extraction techniques were used to address the limitations of the MV techniques developed so far.

The literature survey, the theory and the various concepts of the corner detection techniques, along with a detailed explanation of the two corner detection techniques, being considered in this effort, are presented in Chapter 2. The AAR simulation scheme along with the simulations developed in Simulink is presented with details in Chapter 3.

The UAV software, which consists mainly of the MV algorithms, is presented with details in Chapter 4. Experimental Results and Discussions are presented in Chapter 5. This document is concluded with suggestions for future work in Chapter 6.

Chapter 2 Literature Review

2.1 Feature Extraction

Feature extraction is a specific area of image processing, which involves the use of algorithms for detecting and isolating different desired portions or shapes (features) of a digitized image or a video stream. These features are used to extract information about the image, to compare and match images, and to detect moving objects in a video stream etc. Feature extraction usually deals with small number of well-defined image features i.e. low-level descriptors, such as corners or edges, or high-level descriptors such as basic matching entities [25, 26]. Low-level descriptors can be further classified into three main groups:

- Zero dimensional feature detection, corresponding to smooth surfaces regions, called *region-based feature detection.*
- One-dimensional feature detection, corresponding to regions where significant intensity variation occurs in one direction, called *edge based feature detection.*
- Two-dimensional feature detection, corresponding to regions where significant intensity variation occurs in both the directions, called *corner-based feature detection.*

This section is entirely devoted to *corner-based feature detection* techniques, since they form the basis of the feature extraction methods in this research approach.

2.2 Corners and Interest Points

Many image-processing applications require the comparison of two images to extract information, detect motion, and track objects. This comparison can be done either by comparing every pixel in the images, which is computationally prohibitive in most of the applications, or by matching only points that are in some way interesting. These points are referred to as interest points and comparison of these points reduces the computation time drastically [27,28]. Many different interest point detectors have been proposed with a wide range of definitions about an interest point. Points of interest can be points of high local symmetry, areas of highly varying texture, or just corner points. Corner points are the intersection points of two or more edges formed between two different objects or parts of the same object.

Corners are local image features formed at the boundaries between two brightness regions, with sufficiently high boundary curvature [25,26]. Corners can also be defined as local image features characterized by locations with high intensity variations in both the x and y directions [29]. The corners have the advantage of being discrete and distinguishable making them easily detectable over time in a sequence of images. Listed below are few approaches, which make use of these points of interest [26,27,30].

- Automate object tracking
- Point matching
- Motion based segmentation
- Recognition
- 3D object reconstruction
- Robot navigation
- Image retrieval and indexing

For optimal corner detection any good corner detector should satisfy the following criteria [28,30, 31]:

- All true corners should be detected;
- No false corners should be detected;
- Corner points should be well localized;
- Detectors should have good stability;
- Detectors should be robust with respect to noise;
- Detectors should be computationally efficient.

The true corners are the real corners of an object or structure within an image. False corners are points, which are detected as corners but are not real corners. The detection of all true corners with no false corners is entirely application dependent, since no detector provides an exact definition of a grayscale corner.

The term 'localization' refers to the accuracy in detecting the corner positions. Figure 2.1 show's an example of good and poor localization. Although good localization is desirable in all applications, it is not critical in applications, which can still produce results with approximated corner positions [27,31].

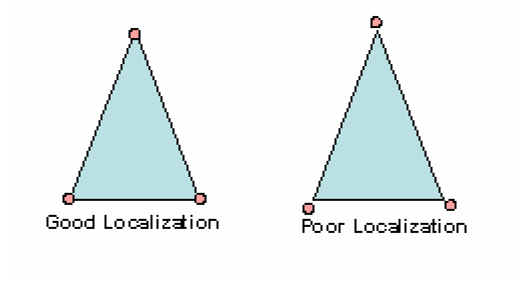


Figure 2.1: Example of good and poor localization

Good stability is defined as the characteristic of the detector to detect a corner in each frame of a video stream. When considering two consecutive images of a video stream they could either be similar or differ by a slight geometric, illumination, or viewpoint transformation. A corner detector that is robust against such transformation is said to have good "stability" [27].

Noise in image is unavoidable in most of the applications. Noises arise as a result of variation in the detector sensitivity, variation in environmental conditions, transmission etc, and in many cases reduces the image quality [32]. Corner detectors are robust to noise, only when they do not detect noise as corners. Since most of the applications are real time applications the corner detector should be computationally efficient to run in real time.

Before discussing the different corner detection methods, certain terms need to be defined for a better understanding of the methods:

Cornerness Map:

A cornerness map is obtained after applying the corner detector to the input image. The corner detector calculates a cornerness measure for each pixel which is just a number describing the degree to which the corner detector believes the point to be a corner [25,27].

Thresholding:

Thresholding is done to avoid reporting all the local maxima with very small cornerness values as corners. The cornerness values, which are less than a certain threshold, are set to zero. The threshold value is entirely application dependent [33,34].

Non-maximal Suppression:

For each point in the cornerness map with a threshold, the cornerness value is set to zero if it is less than the cornerness value of all the points within the mask [33,34].

2.3 Review of Different Corner Detector

Since the first corner detector developed in 1970s (Figure 2.2), several methods have been proposed for extracting two-dimensional features in images. Majority of the corner detectors are usually interest point detectors as they assign a cornerness value to each pixel within an image, though differing in the ways of computing the cornerness measure. Discussed here are few of the most prevalent corner detectors.

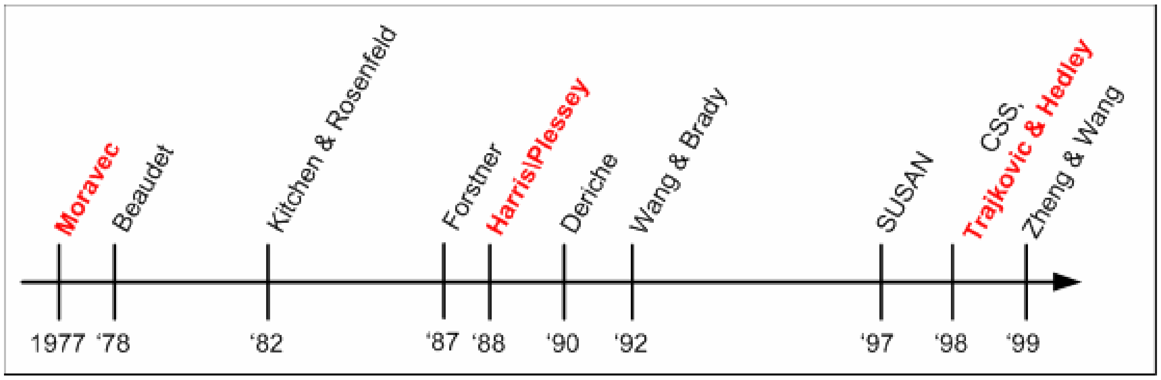


Figure 2.2: A timeline showing the most prominent corner detection techniques [27]

One of the first corner detectors to introduce the concept of ‘points of interest’ in an image was proposed by Moravec [35]. It gave rise to the concept of detecting points of interest within regions of high multi-directional intensity variations. The operator considers a local window in the image and determines the average change in the intensity, resulting from shifting the window by a small amount in various directions. This operation is performed at each pixel and is assigned an interest value equal to the minimum change produced by the shifts. The final response is obtained after thresholding and performing the local non-maximal suppression. The shifts were computed as the non-normalized local autocorrelation functions in four principle directions. Considering only the four principle directions in computing the local autocorrelation makes the approach sensitive to noise. The cornerness value was assigned as the minimum of the autocorrelation function, instead of the variation, which made the approach sensitive to noise along strong edges.

The concept of applying differential geometry operators to corner detection was first proposed by Kitchen and Rosenfeld [36]. The surface parameters were used to find the gradient magnitude and the rate of change of gradient direction (second order derivatives) along an edge contour. The cornerness function of each pixel is defined as the product of the gradient magnitude and the rate of change of gradient direction. Corners are identified by the local maximum of the cornerness measure. This corner detector suffers from sensitivity as it relies on the second order derivative terms and has been shown to have a poor localization of the corners.

Another popular corner detector proposed by Wang and Brady [37] also makes use of the differential geometry operators to detect corners resulting in simplification of the cornerness measure, which is best suitable for real time applications. The Wang and Brady Corner Detector not only requires that the curvature be maximum and above a threshold, but it also requires that the gradient perpendicular to the edge be a maximum and above a threshold. False corner suppression is performed to prevent corners being reported at strong edges. The corners are found at different smoothing levels to ensure an estimation of the corner position without smoothing.

The Beaudet corner detector [38] was one of the first corner detectors to be developed. In this approach, image Gaussian curvature (product of the two principal curvatures) was calculated to enhance the high curvature edges (i.e., detecting the saddle points in the image brightness surface). The cornerness function is given by the determinant of the Hessian matrix. Since the approach involves computation of second-order derivatives it is fairly sensitive to noise.

Deriche and Giraudon [39] proposed methods to obtain accurate corner localization. Corners are found at two different scales; lines are drawn between the two scales for each corner and the intersection of this line with the nearest zero crossing of the Laplacian edge is defined as the correct corner position. Though the approach proposes a new localization technique, it is not clear whether sensitivity to noise and reliability of detection is improved with this method.

Harris and Stephens [40] addressed most of the limitations of the Moravec's operator. The corner detector is built on similar ideas to the Moravec's operator, but the measurement of the local autocorrelation is estimated from first order image derivatives. The variation of the autocorrelations over different directions can be calculated from the principle curvatures of the local autocorrelation. The response is theoretically isotropic, but is often calculated in a way, which makes the response anisotropic. This well conditioned algorithm gives robust detection, i.e. feature points are reliably detected and the detector shows good stability. Comparisons between several algorithms [43,44] have shown that the Harris corner detector reaches the best repeatability rate for moderate changes of the imaging conditions. Furthermore, it was proved that the interest points extracted with the Harris detector has high information content and high saliency [44-50] Even though the operator suffers from poor localization at certain junction types, and the method is computationally expensive, it is still the most widely used corner detector. Nobel [42] proposed a new cornerness measure, which enhances the performance of the Harris detector and is often referred to as the Harris corner detector.

Zheng and Wang [41] proposed a computationally simplified cornerness measure addressing the computational complexity of the Harris operator. The cornerness function was developed based on the Harris operator, identifying the key aspects responsible for corner detection. This corner detector reduces the computational complexity and improves the localization problem, but the performance is slightly degraded with respect to corner detection.

Smith and Brady [31] proposed a novel corner detector, which does not make any assumptions about the form of the localized image structure around a well-localized point; instead it is based on the brightness comparisons within a circular mask. SUSAN (Smallest Univalue Segment Assimilating Nucleus) corner detector assumes that within a small circular region, pixels belonging to a given object will have relatively uniform brightness. The algorithm computes the number of pixels with the same brightness as that of the nucleus. This set of pixels is called USAN (univalue segment assimilating nucleus) of the mask. The mask is applied at each pixel in the image and the corners are detected by finding the local minima in the USAN map, i.e. the USAN area should be less than half the maximum possible area n_{max} for a corner to be present at that point.

The Harris corner detector was selected for this study because of the advantages the detector has over other corner detectors, which operate, on the same autocorrelation principle [44-50]. The SUSAN corner detector was selected for this study because of the detector's different approach to corner detection methods and also because the detector operates without the calculation of the autocorrelation function. Hence, the two chosen corner detectors feature entirely different detection methods.

2.4 Moravec's Interest Point Detector

Moravec's Interest Point Detector considers a local window in the image and determines the average change in the intensity, resulting from shifting the window by a small amount in various directions. There are three distinct cases to be considered, regarding the shifts:

- If the image patch under the window is flat (i.e. constant intensity), then all the shifts will result in only a small change.
- If the image patch under the window is an edge, then a shift along the edge will result in a small change but a shift perpendicular to the edge will result in a large change

- If the patch under the window is a corner then a shift in any direction will cause a large change. Hence a corner is detected when a minimum change in the shifts produces a large change

The mathematical expression for the above-mentioned shifts is given as:

$$E_{x,y} = \sum_{u,v} w_{u,v} \mid I_{x+u,y+v} - I_{u,v} \mid^2 \tag{2.1}$$

where *w* specifies the image window which is unity within the rectangular regions and zero else where. E is the change in the intensities produced due to the shifts {x, y}. The considered shifts {x, y} comprise the {(1, 0), (1, 1), (0, 1), (-1, 1)}. This operation is performed at each pixel position and is assigned a cornerness value equal to the minimum of the measurement E, which is the minimum change produced by the shifts. The points of interest are the local maxima points of the cornerness map.

The response is anisotropic because only a discrete set of shifts at every 45 degrees is considered. The response is also noisy because of the binary and rectangular window function. Since the corner measure is only the minimum of the measurement E, the operator responds too readily to edges. These three drawbacks were addressed by the Harris corner detector.

2.5 Harris Corner Detector

Harris corner detector proposed a method to consider all possible small shifts instead of just the shifts at every 45 degrees, by performing the analytic expansion of Equation 2.1 about the shift origin:

$$\begin{aligned} E_{x,y} &= \sum_{u,v} w_{u,v} \left[I_{x+u,y+v} - I_{u,v} \right]^2 \\ &= \sum_{u,v} w_{u,v} \left[xX + yY + O\{x^2, y^2) \right]^2 \end{aligned} \tag{2.2}$$

where the first gradients are approximated by

$$X = I \otimes (-1,0,1) = \frac{\partial I}{\partial x}$$
$$Y = I \otimes (-1,0,1)^T = \frac{\partial I}{\partial y} \tag{2.3}$$

Hence, the expression for all possible small shifts, can be rewritten as

$$E(x,y) = Ax^2 + 2Cxy + By^2 \tag{2.4}$$

where

$$A = X^2 \otimes w$$
$$B = Y^2 \otimes w \tag{2.5}$$
$$C = \{XY\} \otimes w$$

Harris corner detector proposed the use of a smooth circular Gaussian window instead of the binary and rectangular window to overcome the noisy response of the Moravec's operator. The cornerness function has been modified to make the corner detector insensitive to edges. The cornerness function was reformulated making use of the variation in E along with the direction of the shift. The change E for a small shift (x, y) is given as

$$E(x,y) = (x,y)M(x,y)^T \tag{2.6}$$

where the 2x2 symmetric matrix M, is given as

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{2.7}$$

The function E is closely related to the local auto-correlation function. Let α and β be the eigen values of the M matrix. The α and β will be proportional to the principle curvature of the local autocorrelation function and form a rotationally invariant description of M. Based on the values of α and β three different cases should be considered:

- If both the curvatures are small, the autocorrelation function is flat, and the windowed region is approximately of constant intensity.

- If the curvatures are alternately high and low, the autocorrelation function is ridge shaped, and the shifts along the ridge causes little change in E indicating an edge.
- If both the curvatures are high, the autocorrelation function is sharply peaked and the shifts in any direction will increase E, indicating a corner.

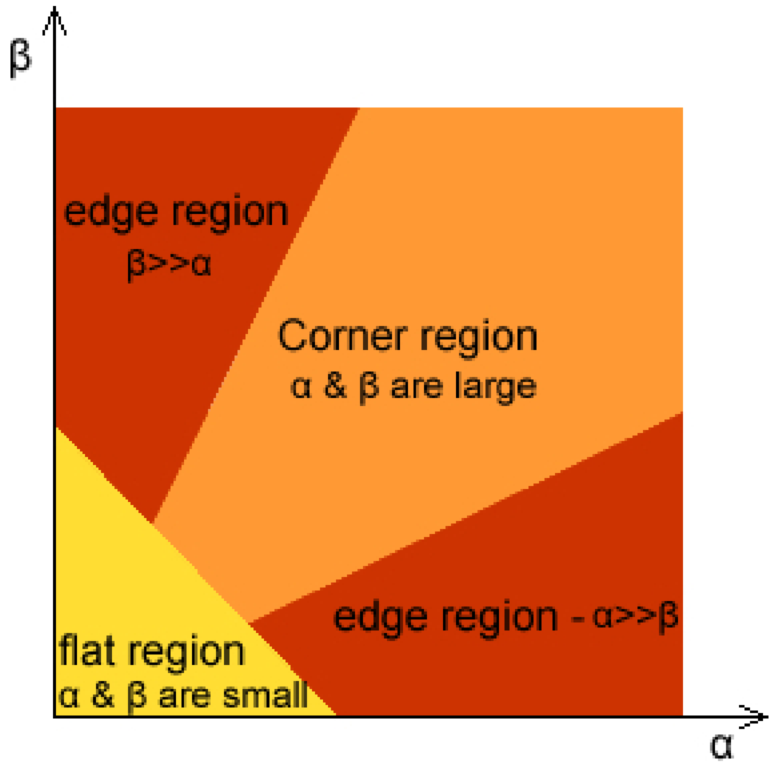


Figure 2.3: Autocorrelation principle curvature plane with the corner/edge/flat region classification

Figure 2.3 describes the (α , β) space. An ideal edge will have α large and β zero, but in reality the value of β is never zero, because of the noise, intensity quantization, and pixellation. Both α and β being large, indicates a corner and both of them being small indicates a flat region.

To ensure that all the detected points, which fall within the classification region, are really corners, a cornerness measure is specified which determines the quality of the detected points. The Harris corner detector's cornerness function is given by:

$$C = Det(M) - k(Trace(M))^2 \tag{2.8}$$

where *k* is a constant which is generally assumed to be 0.04, and the larger the value of *k* the less sensitive is the detector to corner like structures Finally a non maximum suppression is performed to determine the final corners. The drawback of this cornerness function is the value *k* as it needs to be tuned manually. A modified cornerness function to overcome this problem was proposed by Noble [21] and is given as

$$C = \frac{\det(M)}{Tr(M) + \varepsilon} \tag{2.9}$$

The constant ε is used to avoid singular denominator in case of a rank zero autocorrelation matrix (M).

2.6 SUSAN Principle

The SUSAN corner detector describes an entirely new approach to the low-level image processing, specially the edge and corner detection. Consider Figure 2.4, which shows the circular mask and a simple image of a rectangular block. The mask, whose center pixel is called the nucleus, is placed at four different positions on the block. The brightness or intensity of each pixel within the mask is compared with that of the mask's nucleus and an area of the mask is defined which has the same/similar brightness as the nucleus and assigned as a value to the pixel.

This area of similar brightness is known as the USAN an acronym standing for "Univalue Segment Assimilating Nucleus". This concept of each image point having associated with it a local area of similar brightness is the basis of the SUSAN principle.

In Figure 2.4 the USAN for the four mask positions is shown in pink. The area of the USAN conveys the most information. The USAN area is at a maximum when the nucleus is on a flat surface (positions b and c in Fig b of Figure 2.4), and the USAN is half of this maximum when the nucleus is near an edge (position d in Fig b of Figure 2.4) and the USAN area decreases further when the nucleus is at a corner (position a in Fig b of Figure 2.4). Hence it is USANs area which is used to determine the two dimensional features and edges or in other words the smallest USAN would give us the two dimensional feature. Hence the term SUSAN standing for "Smallest Univalue Segment Assimilating Nucleus".

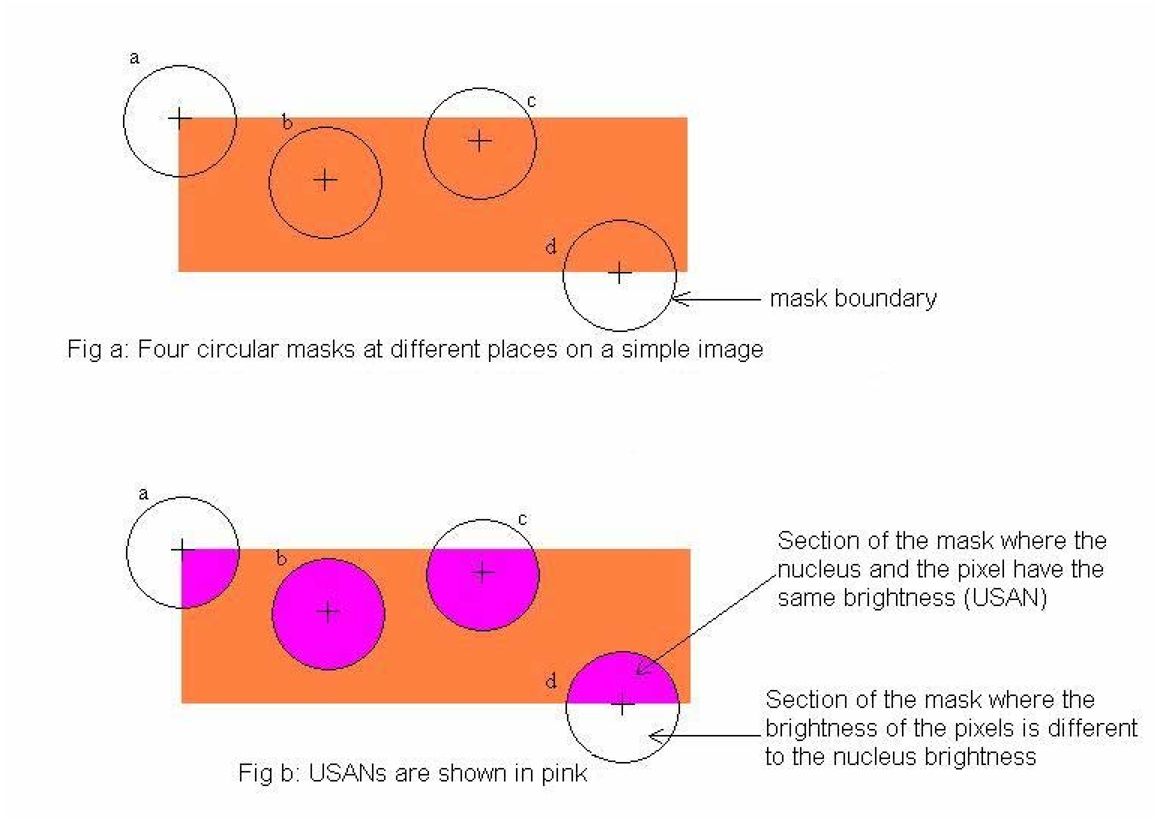

a
b
c
d
mask boundary
Fig a: Four circular masks at different places on a simple image
a
b
c
d
Section of the mask where the nucleus and the pixel have the same brightness (USAN)
Section of the mask where the brightness of the pixels is different to the nucleus brightness
Fig b: USANs are shown in pink

Figure 2.4: SUSAN Principle [31]

2.7 SUSAN Corner Detector

The SUSAN corner detector [31] was implemented using a circular mask of 37 pixels to calculate the USAN area. The mask is placed at each pixel in the image and the brightness of each pixel within the mask is compared with that of the nucleus based on the equation:

$$c(\vec{r},\vec{r}_0) = e^{-\left(\frac{I(\vec{r})-I(\vec{r}_0)}{t}\right)^6} \tag{2.10}$$

where $\vec{r}_0$ is the position of the nucleus in the image, $\vec{r}$ is the position of any other pixel within the mask, $I(\vec{r})$ is the brightness of any pixel within the mask, t the brightness threshold, and c is the output of the comparison. The area of the USAN is obtained by summing up the outputs c.

$$n(\vec{r}_0) = \sum_{\vec{r}} c(\vec{r}, \vec{r}_0) \tag{2.11}$$

where n is the USAN area. If the nucleus is on a corner then the USAN area n will be less than half of the mask area $n_{\max}$ and will be a local minimum. To determine if the area is less than half, the value n is compared with the geometric threshold g which is set to be equal to exactly half the $n_{\max}$. Based on the 37 pixel mask and the Eq (2.10) and (2.11), the $n_{\max}$ value was calculated to be 37 and hence the value of g to be18.50.

The geometric threshold g affects the number of corners detected and, more importantly, the shape of these corners. The corners would be much sharper if the geometric threshold g is reduced. The brightness threshold t on the other hand affects the number of corners detected. Since the brightness threshold determines the allowed brightness variation within the USAN, a reduction in this value, makes the detector sensitive to subtle variations in the image leading to more number of corners detected. Finally, non-maximum suppression is performed to suppress corners with USAN area less than the USAN area of the pixels in the neighborhood of the non-maximal suppression mask.

The SUSAN corner detector has the advantage of being robust to noises and yield accurate outcomes along with a reasonable computation speed. However the algorithm may generate false corners when operated on low contrast images, or blurred images [51].

Chapter 3 Experimental Setup

3.1 The AAR Simulink Simulation Scheme

A sketch of the Autonomous Aerial Refueling (AAR) system is shown in Figure 3.1. The relevant reference frames, the problem formulation, sensors, and distance vectors will be described in this section.

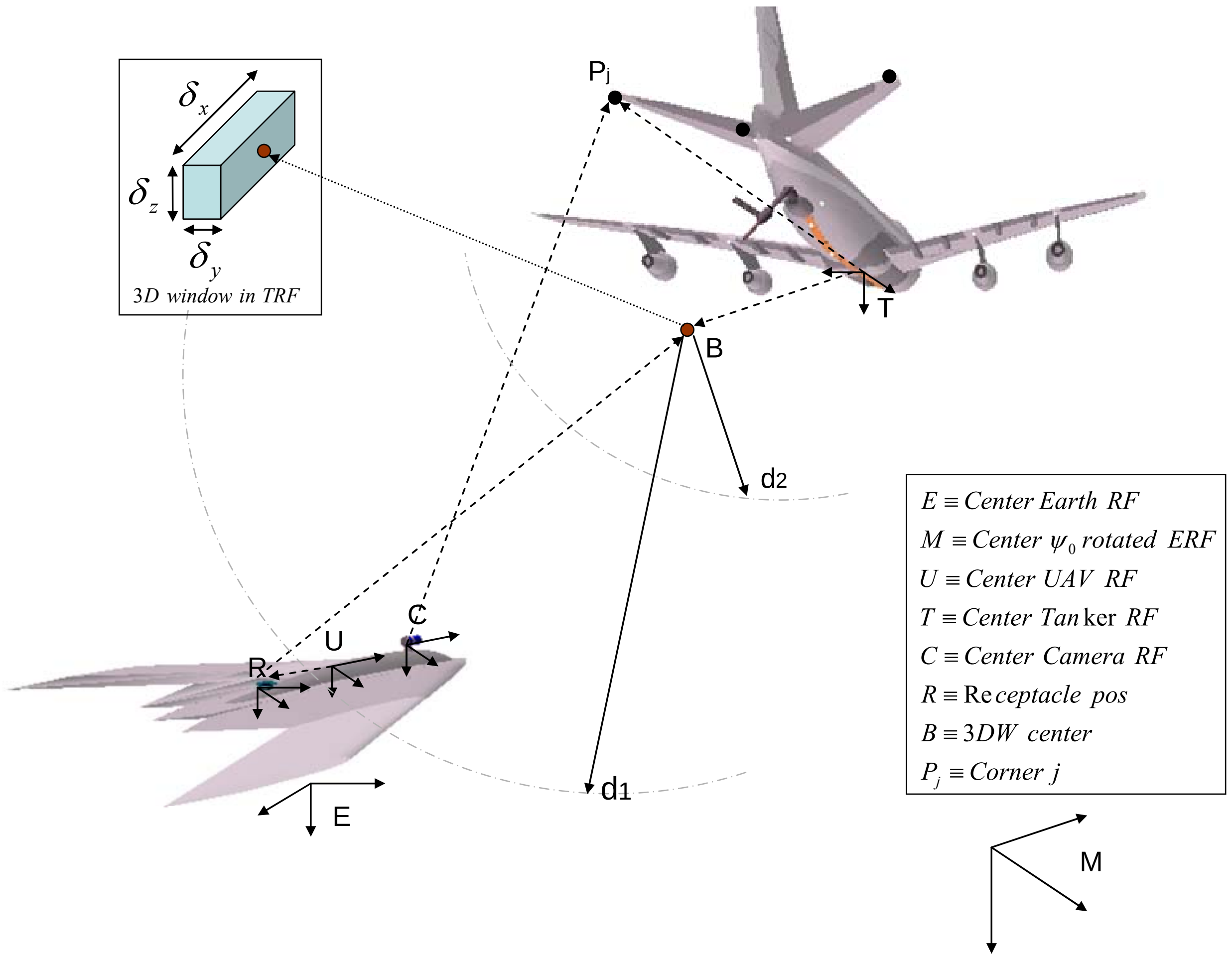

δ_x
δ_z
δ_y
3D window in TRF
P_j
T
B
d_2
d_1
C
U
R
E
M
$E \equiv$ Center Earth RF
$M \equiv$ Center ψ_0 rotated ERF
$U \equiv$ Center UAV RF
$T \equiv$ Center Tanker RF
$C \equiv$ Center Camera RF
$R \equiv$ Receptacle pos
$B \equiv$ 3DW center
$P_j \equiv$ Corner j

Figure 3.1: Reference Frames of the AAR problem

3.1.1 Reference Frames

The study of the AAR problem requires the definition of specific Reference Frames (RFs). The respective reference frames are shown in Figure 3.1.

- ERF: Earth fixed Reference Frame
- TRF: Tanker body fixed Reference Frame
- URF: UAV body fixed Reference Frame
- CRF: Fixed UAV Camera Reference Frame

The TRF and the URF are located at the center of gravity of the aircraft. To make the docking problem invariant with respect to the nominal heading of the aircraft, an additional fixed frame MRF is defined and is rotated by the nominal heading angle ψ_0 with respect to the ERF.

Within this thesis, the following notation has been used:

- Geometric points are expressed using the homogenous (4D) coordinates and are denoted with a capital letter and a left superscript indicating the reference frame in which the point is expressed. For example, a point *P* expressed in the *F* reference frame, has coordinates ${}^{F}P = [x,y,z,1]^T$, (where the right *'T'* superscript indicates transposition).
- Vectors are defined as difference between points; therefore, their 4th coordinate is always '0'. Also, vectors are denoted by two uppercase letters, indicating the two points at the extremes of the vector; for example, ${}^{E}BR = {}^{E}B - {}^{E}R$ is the vector from the point R to the point B, expressed in the Earth Reference Frame.
- Transformation matrices are (4 x 4) matrices that transform points and vectors expressed in an initial reference frame to points and vectors expressed in a final reference frame. They are usually denoted with a capital *T*, with a right subscript indicating the "initial" reference frame and a left superscript indicating the "final" reference frame. For example the matrix ${}^{E}T_{T}$ represents the homogeneous transformation matrix that transforms a vector/point expressed in TRF to a vector/point expressed in ERF.

3.1.2 Geometric Formulation of the AAR problem

The objective is to guide the UAV such that its fuel receptacle (i.e. point *R* in Figure 3.1) is transferred to the center of a 3-dimensional window (3DW) under the tanker (point *B*). Once the UAV fuel receptacle reaches and remains within this 3DW, it is assumed that the boom operator can take control of the refueling operations. It should be emphasized that point *B* is fixed within the TRF, and that the dimensions of the 3DW $(\delta x, \delta y, \delta z)$ are a design parameter. Table 3.1 specifies the desired and allowable limits of the 3DW dimensions. These values were selected from publicly available images of the aerial refueling for manned aircrafts.

	Desired (meter)	**Limit (meter)**
δx	±0.40	±2.10
δy	±1.87	±2.10
δz	±0.90	±2.56

Table 3.1: Dimension specification of the 3D refueling window

3.1.3 Distance Sensors

It is assumed that the tanker and the UAV can share a short-range data communication link during the docking maneuver. Both the UAV and the tanker possess GPS systems. Furthermore, it is assumed that the UAV is equipped with a digital camera along with an on-board computer hosting the MV algorithms that acquires the images of the tanker. Finally, the 2-D image plane of the MV system is defined as the '*y-z*' plane of the CRF.

3.1.4 Receptacle 3D-window center vector

The reliability of the AR docking maneuver is based on the accuracy of the measurement of the vector ${}^{T}RB$, which is the distance between the UAV fuel receptacle and the center of the 3D refueling window, expressed in TRF:

$$ {}^{T}RB = {}^{T}B - {}^{T}T_{U}\,{}^{U}R \tag{3.1} $$

where ${}^{T}T_{U} = \left({}^{U}T_{C}\,.\,{}^{C}T_{T}\right)^{-1}$ and ${}^{C}T_{T} = {}^{C}T_{U}\left({}^{E}T_{U}\right)^{-1}\,{}^{E}T_{T}$. Since the fuel receptacle and the 3DW center are located at fixed and known positions with respect to center of gravity of the UAV and tanker respectively, both ${}^{U}R$ and ${}^{T}B$ are known and constant. The matrix ${}^{C}T_{U}$ expresses the position and attitude of CRF with respect to the URF, and therefore is also known and generally constant. The transformation matrix ${}^{C}T_{T}$ can be evaluated either “directly”- that is using the relative position and orientation information provided by the MV system- or “indirectly”- that is by using the matrices ${}^{E}T_{U}$ and ${}^{C}T_{T}$, which in turn can be evaluated using information from the position and attitude sensors of the tanker and UAV respectively.

3.2 The AR Simulation Environment

The AR simulation scheme was developed using Simulink®. The simulation outputs were linked to a Virtual Reality Toolbox (VRT) interface to provide typical scenarios associated with the AR maneuvers. The interface allows the positions of the simulated objects such as the UAV and tanker, to drive the position and orientation of the corresponding objects in a Virtual World. Figure 3.2 shows the AAR simulation scheme developed at WVU. Several objects including the tanker, the landscape, and different parts of the boom were originally modeled using 3D Studio and later exported to Virtual Reality Modeling Language (VRML). Every object was scaled according to its real dimensions. Different viewpoints were made available to the user, including the view from the UAV camera and the view from the boom operator Figure 3.3 shows the viewpoints from the UAV camera and the viewpoint from the boom operator. The simulation main scheme features a number of graphic user interface (GUI) menus allowing the user to set a number of simulation parameters including:

- Initial position of the UAV with respect to the tanker;

- Level of atmospheric turbulence, sensor and GPS noise;
- Location of the camera on the UAV and its orientation within the UAV body frame;
- Location of the fuel receptacle on the UAV;
- The number of corners and the location of physical corners of interest (Figure 3.4).

From the Virtual World environment, images of the tanker as seen from the UAV camera are continuously acquired and processed during the simulation.

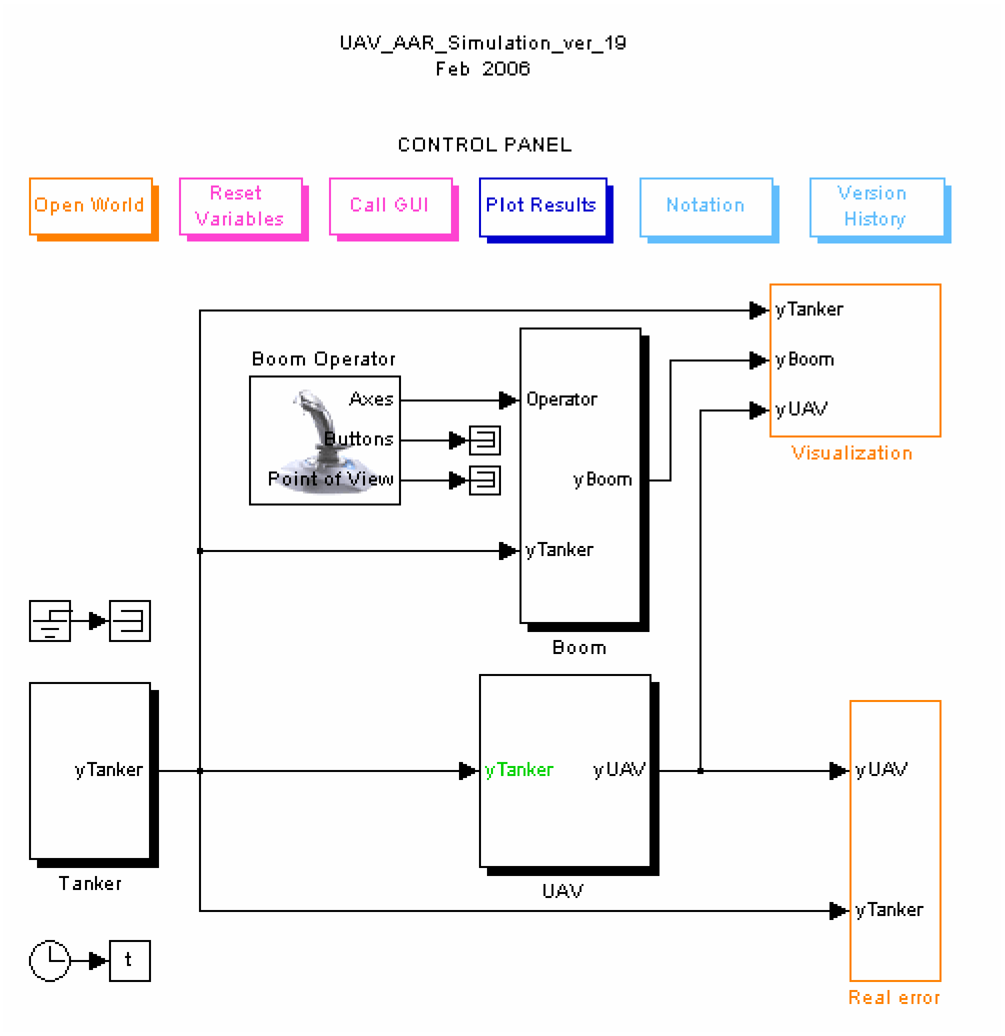
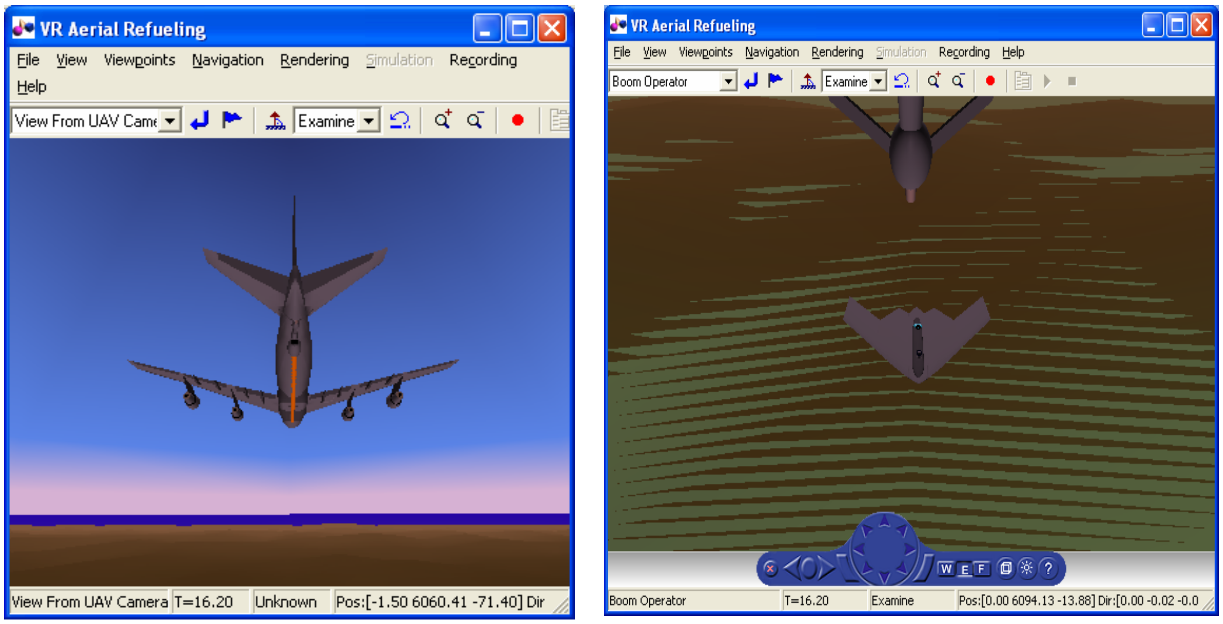


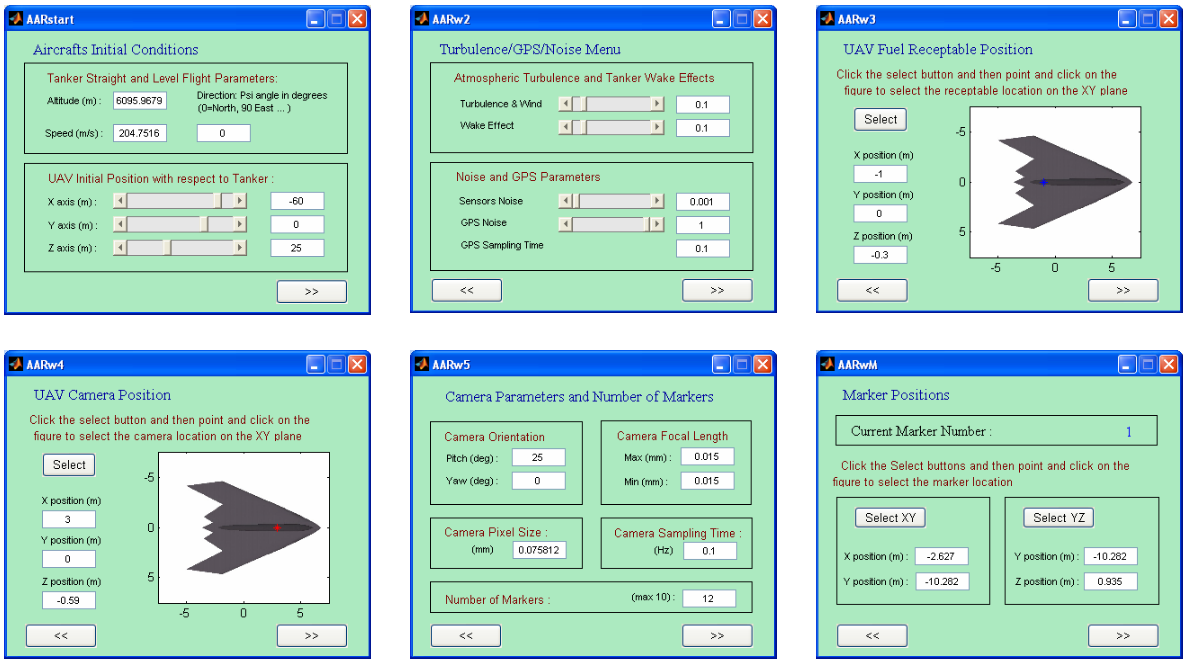
Figure 3.2: Simulink model of the AAR simulation scheme developed at WVU


VR Aerial Refueling
File View Viewpoints Navigation Rendering Simulation Recording Help
View From UAV Came
Examine
VR Aerial Refueling
File View Viewpoints Navigation Rendering Simulation Recording Help
Boom Operator
Examine

View From UAV Camera T=16.20 Unknown Pos:[-1.50 6060.41 -71.40] Dir

Boom Operator T=16.20 Examine Pos:[0.00 6094.13 -13.88] Dir:[0.00 -0.02 -0.0

Figure 3.3: The Virtual Reality windows from the AAR simulation



AARstart

Aircrafts Initial Conditions

Tanker Straight and Level Flight Parameters:

Altitude (m) : 6095.9679 Direction: Psi angle in degrees (0=North, 90 East ...)

Speed (m/s) : 204.7516 0

UAV Initial Position with respect to Tanker :

X axis (m) : -60

Y axis (m) : 0

Z axis (m) : 25

AARw2

Turbulence/GPS/Noise Menu

Atmospheric Turbulence and Tanker Wake Effects

Turbulence & Wind 0.1

Wake Effect 0.1

Noise and GPS Parameters

Sensors Noise 0.001

GPS Noise 1

GPS Sampling Time 0.1

AARw3

UAV Fuel Receptable Position

Click the select button and then point and click on the figure to select the receptable location on the XY plane

X position (m) -1

Y position (m) 0

Z position (m) -0.3

AARw4

UAV Camera Position

Click the select button and then point and click on the figure to select the camera location on the XY plane

X position (m) 3

Y position (m) 0

Z position (m) -0.59

AARw5

Camera Parameters and Number of Markers

Camera Orientation

Pitch (deg) : 25

Yaw (deg) : 0

Camera Focal Length

Max (mm) : 0.015

Min (mm) : 0.015

Camera Pixel Size : (mm) 0.075812

Camera Sampling Time : (Hz) 0.1

Number of Markers : (max 10) : 12

AARwM

Marker Positions

Current Marker Number : 1

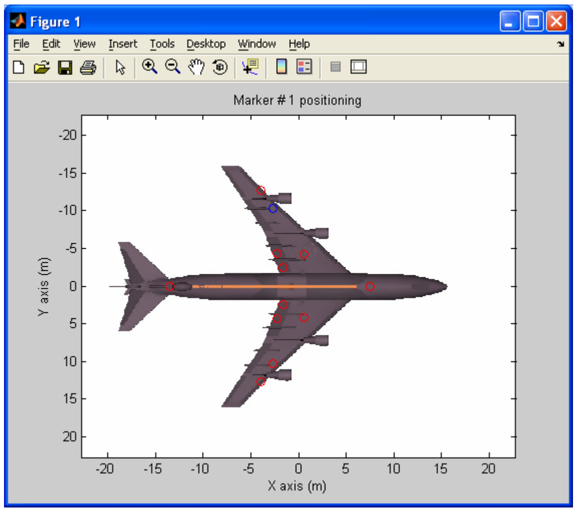
Click the Select buttons and then point and click on the figure to select the marker location

X position (m) : -2.627

Y position (m) : -10.282

Y position (m) : -10.282

Z position (m) : 0.935



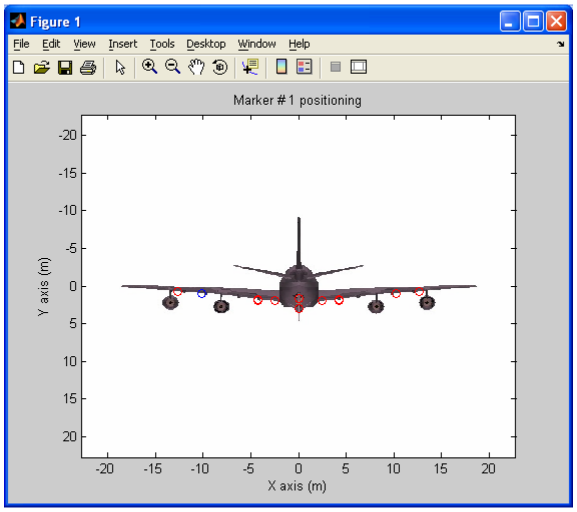
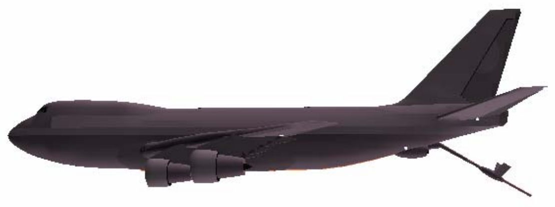


Figure 3.4: Graphical User Interfaces

3.2.1 Tanker

The graphic tanker model used in the AAR simulation is a B747 model. The graphic model was re-scaled to match the size of a KC-135 tanker. The KC-135 Stratotanker's primary mission is to refuel long-range aircrafts. The Simulink model of the tanker also accommodates the GPS sensors and the boom.



(a): Tanker from AAR simulation

(b): KC 135R Tanker

Figure 3.5: The Virtual Reality tanker model and the real KC-135R model

3.2.1.1 Modeling of the Tanker System

The Simulink tanker model consists of the general aircraft model, actuator dynamics, and the sensors. Figure 3.6 shows the Simulink model of the Tanker.

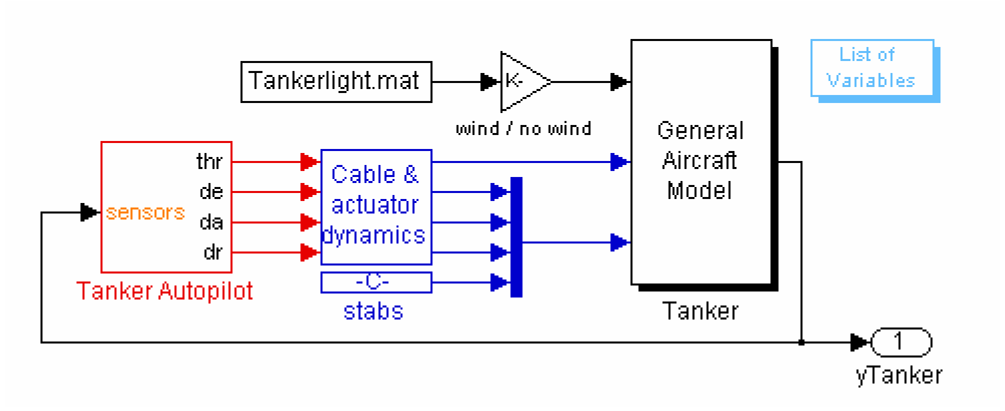


Figure 3.6 Simulink model of the tanker system

The tanker model has the dynamic characteristics of the Boeing KC-135R and was modeled with the parameters specified in Ref.52. The aircraft assumes a steady state equivalent to a rectilinear trajectory; a constant Mach number of 0.65 and an altitude (H) of 6000m.The lateral dynamic motion were eliminated by limiting the aircraft to just longitudinal motion. The longitudinal motion has stable dynamics and the tanker does not require an internal stability control. The non-linear aircraft models of the tanker have been developed using the conventional modeling procedures and conventions [53].

The state vector $x = [V,\alpha,\beta,p,q,r,\psi,\theta,\phi,x,y,z]^T$ describes the 12-state model of the tanker, where the first six variables are expressed within the body reference frame and the last six are in the earth reference frame (ERF). First order responses have been assumed for the actuator dynamics using typical values for aircraft's of similar size and/or weight. The tanker autopilot system is designed using LQR based control laws.

3.2.1.2 Modeling of the Boom

The simulation includes a detailed modeling of the elastic behavior of the boom [16]. The boom has been modeled using the Finite Element Model (FEM) scheme represented in Figure 3.7. The boom is connected to the tanker at point *P* and consists of two elements; the first element is connected to point *P* by two revolute joints, allowing vertical and lateral relative rotations $(\theta_4 \, and \theta_5)$; the second element is connected to the first one by a prismatic joint that allows the extension d_6.

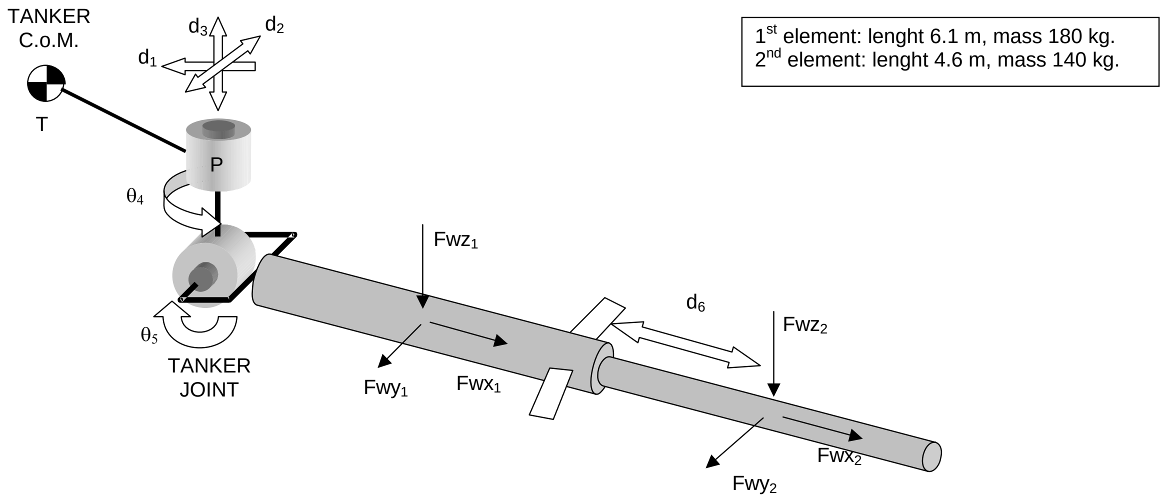

TANKER
C.o.M.
T
d3
d2
d1
P
θ4
θ5
TANKER
JOINT
Fwz1
Fwy1
Fwx1
d6
Fwz2
Fwx2
Fwy2
1st element: lenght 6.1 m, mass 180 kg.
2nd element: lenght 4.6 m, mass 140 kg.

Figure 3.7: Model of the Refueling boom

The dynamic modeling of the boom has been derived using the Lagrange method:

$$\frac{d}{dt}\frac{\partial L(q,\dot{q})}{\partial \dot{q}_i} - \frac{\partial L(q,\dot{q})}{\partial q_i} = F_i \qquad i = 1,2,....,n \tag{3.2}$$

where $L(q,\dot{q}) = T(q,\dot{q}) - U(q)$ is the Lagrangian (difference between the boom kinetic and potential energy), q is the vector of the Lagrangian coordinates and the F_i are the Lagrangian forces on the boom. To derive the Lagrangian, reference is made to the ERF. The inertial and the gravitational forces are implicitly included in the left hand side of Equation (3.2) and the F_i represents the active forces (wind and control forces). With respect to the ERF, the boom has six degree of freedom: the three translations d_1, d_2, and d_3 of point P, the rotations θ_4 and θ_5, and the extension d_6; therefore the Lagrangian coordinates can be chosen as $q = [d_1, d_2, d_3, \theta_4, \theta_5, d_6]^T$. The first three variables d_1, d_2, and d_3 (the position of point P in a given frame) can be expressed as:

$$
\begin{aligned}
&{}^{E}P(t)={}^{E}T(t)+{}^{E}TP \\
&{}^{E}\dot{P}(t)={}^{E}\dot{T}(t)+\omega\times{}^{E}TP \\
&{}^{E}\ddot{P}(t)={}^{E}\ddot{T}(t)+\dot{\omega}\times{}^{E}TP+\omega\wedge(\omega\wedge{}^{E}TP)
\end{aligned}
\tag{3.3}
$$

Where ${}^{E}T$ is the position of the tanker's center of gravity, ω is the tanker angular velocity, ${}^{E}P=\left[d_1,d_2,d_3\right]^T$, ${}^{E}TP$ is the fixed length vector going from ${}^{E}T$ to ${}^{E}P$. The kinetic and potential energies have been derived referring to the Denavit-Hartenberg representation of the system [54, 55].

	a_i	α_i	d_i	θ_i
1	0	$\frac{\pi}{2}$	d_1	0
2	0	$\frac{\pi}{2}$	d_2	$\frac{\pi}{2}$
3	0	$\frac{\pi}{2}$	d_3	0
4	0	$-\frac{\pi}{2}$	0	θ_4
5	0	$-\frac{\pi}{2}$	0	θ_5
6	0	$\frac{\pi}{2}$	d_6	$\frac{\pi}{2}$

Table 3 2: Denavit Hartenberg boom parameters

3.2.2 Unmanned Aerial Vehicle modeling

The graphic UAV model used in the AAR simulation is a B2 model. The graphic model was re-scaled to match the size of the ICE 101 aircraft. The Simulink UAV model consists of the general aircraft model, sensors, UAV software, and the actuators.

3.2.2.1 Modeling of the UAV System

The Simulink UAV aircraft model was based on the design parameters of an ICE-101 aircraft model [56]. This model was developed using the conventional modeling approach outlined by Ref. 57. A 12 steady state model describes the resulting UAV model:

$$x = \left[V, \alpha, \beta, p, q, r, \psi, \theta, \varphi, {}^{E}x, {}^{E}y, H\right] \tag{3.4}$$

where x is the state variable; V (m/s) is the component x of the velocity in body axis; α (rad) is the wind axis angle of attack; β (rad) is the wind axis sideslip angle; p, q, r (rad/sec) are the components (x, y, z) of the angular velocity in body axis (also known as roll, pitch and yaw rates); ψ, θ, φ (rad) are the yaw, pitch and roll Euler angles; ${}^{E}x$, ${}^{E}y$, H are the position in ERF.

The angle of attack α and the sideslip angle β are defined as:

$$\alpha = \tan^{-1}\left(\frac{W}{\left\|\overline{V}\right\|}\right) \quad \text{and} \quad \beta = \sin^{-1}\left(\frac{U}{\left\|\overline{V}\right\|}\right) \tag{3.5}$$

where $\overline{V} = \left[V, U, W\right]$ is the linear velocity in body axis.

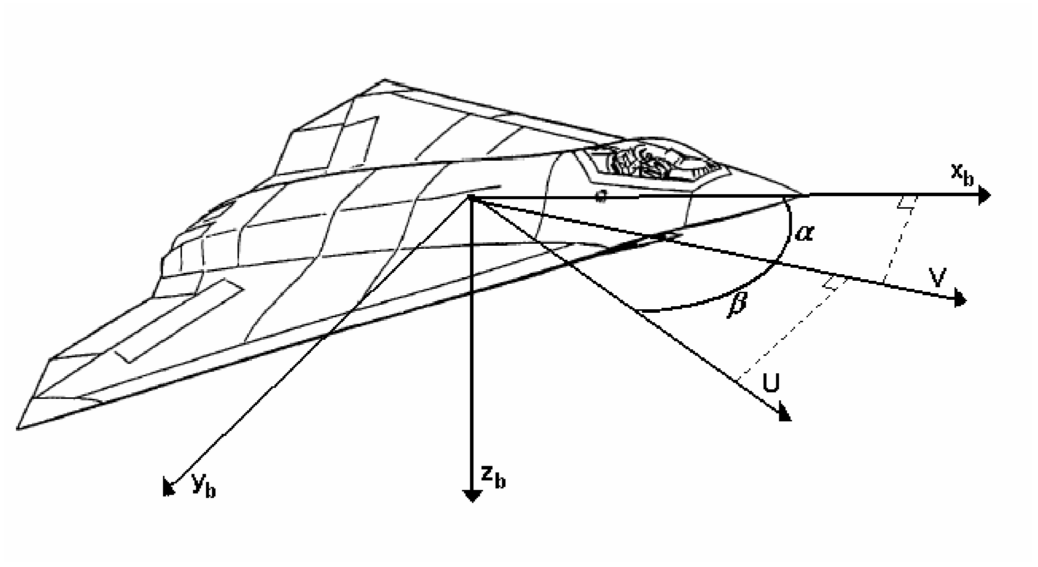


Figure 3.8: Angle of attack α and sideslip angle β of the UAV aircraft

The input vector *u* is:

$$u = \left[\delta_{Throttle}, \delta_{AMT\_R}, \delta_{AMT\_L}, \delta_{TEF\_R}, \delta_{TEF\_L}, \delta_{LEF\_R}, \delta_{LEF\_L}, \delta_{PF}, \delta_{SSD\_R}, \delta_{SSD\_L} \right] \quad (3.6)$$

where AMT is All Moving Tips, TEF is Trailing Edge Flaps, LEF is Leading Edge Flaps, PF is Pitch Flaps, SSD is Spoiler Slot Deflector. These parameters are shown in the Figure 3.9.

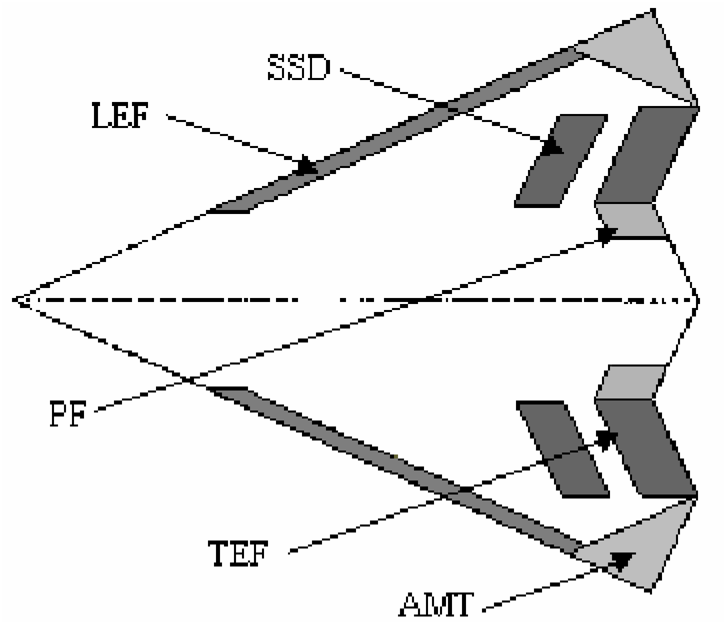


Figure 3.9: Control surfaces of the UAV aircraft

The dynamic UAV model can be described in general by the differential equation

$$\dot{x}(t) = f(x, u, t) \quad (3.7)$$

which can be linearized at a trim point such as:

$$\dot{x}(t) = f(x_0, u_0, t) = 0 \quad (3.8)$$

In the above condition, the UAV has acceleration equal to zero and has the vector velocity constant. The equivalent state space model is an LTI system with 12 state variables as shown in the Equation 3.9.

$$\dot{x} = Ax + Bu \quad (3.9)$$

The analysis of Eq (3.9) shows that there is a substantial decoupling between the longitudinal symmetric motion (translation *x*, translation *z* and rotation *y*) and the lateral-directional asymmetric motion (translation *y*, rotation *x* and rotation *z*).

The longitudinal motion is characterized by 2 modes, the first one with high damping and high frequency (short period) dominated by negligible variation in velocity, the second one with low frequency (phugoid) characterized by small variation in incidence and slow variation in the pitch angles:

$$\begin{aligned} \lambda_1 &= \lambda_{1,R} \pm j\lambda_{1,I} \quad (short\ period) \\ \lambda_2 &= \lambda_{2,R} \pm j\lambda_{2,I} \quad (phugoid) \end{aligned} \tag{3.10}$$

The lateral-directional motion is characterized by 4 real modes, of which 2 are unstable:

$$\begin{aligned} \lambda_3 &> 0 \\ \lambda_4 &< 0 \\ \lambda_5 &< 0 \\ \lambda_6 &> 0 \end{aligned} \tag{3.11}$$

For this reason, the model results with very unstable lateral dynamics.

3.2.2.2 Sensors

A UAV typically requires a larger number of sensors than a normal airplane. The precision in the sensors play a major role in the performance of a system, in a UAV this is even more important since there is no human intervention during the flight phases. It is assumed the UAV has a GPS system designed on the same terms as the tanker, an Inertial Navigation Unit (INU) with gyros and accelerometers along with a MV system.

The INU provides ψ, θ, φ (rad), and p, q, r (rad/sec) data, with some added noise. In this effort, a Band-limited White Gaussian Noise (BWGN) with a power of (n_p) = $1*10^{-9}$ and sample time T =0.05 sec is assumed for the euler angles, and a BWGN with a $n_p = 1*10^{-8}$ and T=0.05 sec, is assumed for $p, q,$ and r. The measurements of the velocity angles α and β (rad) have BWGN with a $n_p = 1*10^{-9}$ and T=0.05 sec, and the velocity V (m/s) has BWGN with a $n_p = 1*10^{-7}$ and T=0.05 sec. The accelerations and the other measurements do not have any added WGN.

The GPS system provides the ${}^{E}x$, ${}^{E}y$, H, to which a noise designed from real experimental data is added. Furthermore, the GPS model features a unit step delay to better approximate the real behavior of this system. The simulation of this sensor is shown in Figure 3.10

A BWGN is a simulation of White Gaussian Noise with the Power Spectral Density (PSD) equal to n_p, the correlation time equal to the sample time T and the covariance as $c = \frac{n_p}{T}$.

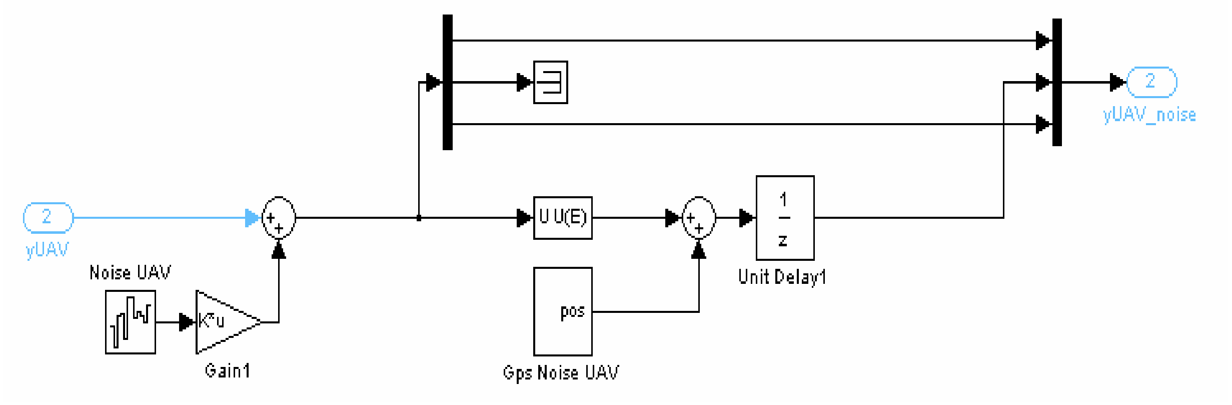


Figure 3.10: Simulink model of the UAV GPS system

The MV system on the other hand can be considered to be a smart sensor that provides the distance between the camera and the tanker.

3.2.2.3 UAV Software

The UAV software consists of the MV block, the fusion block and the controller. The MV system is a critical part of the UAV software and will be explained with details in the following chapter.

3.2.2.4 Atmospheric Turbulence and Wake Effects

The atmospheric turbulence acting on both tanker and the UAV aircraft was modeled using the Dryden wind turbulence model [22]. A ‘light’ turbulence was selected since aerial refueling is typically performed at high altitudes in calm air [22]. The wake effects of the tanker on the UAV are more significant than the atmospheric turbulence and have been modeled through the interpolation of a large amount of experimental data [23, 24] as perturbations to the aerodynamic coefficients $C_D, C_L, C_m, C_l, C_n, C_Y$ for the UAV aerodynamic forces and moments. These coefficients represent drag and lift coefficients, rolling, pitching and yawing moment coefficients and side force coefficient, all of them subject to variations due to formation flight. Figure 3.11 shows the wind tunnel tests to measure close formation aircraft aerodynamic characteristics.

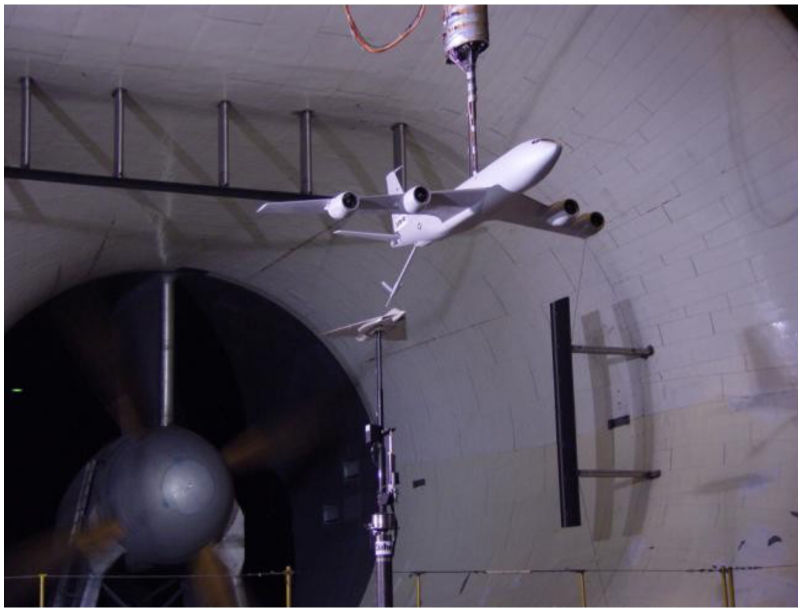


Figure 3.11: Wind tunnel test

3.2.2.5 Actuator Dynamics

Every actuator of the input vector in Equation.3.6 has saturation and a rate limiter (Figure 3.12) that is, an upper, lower and a velocity limit. The input vector is delayed and every actuator filtered. The filters are faster for the control surfaces than for the throttle command.

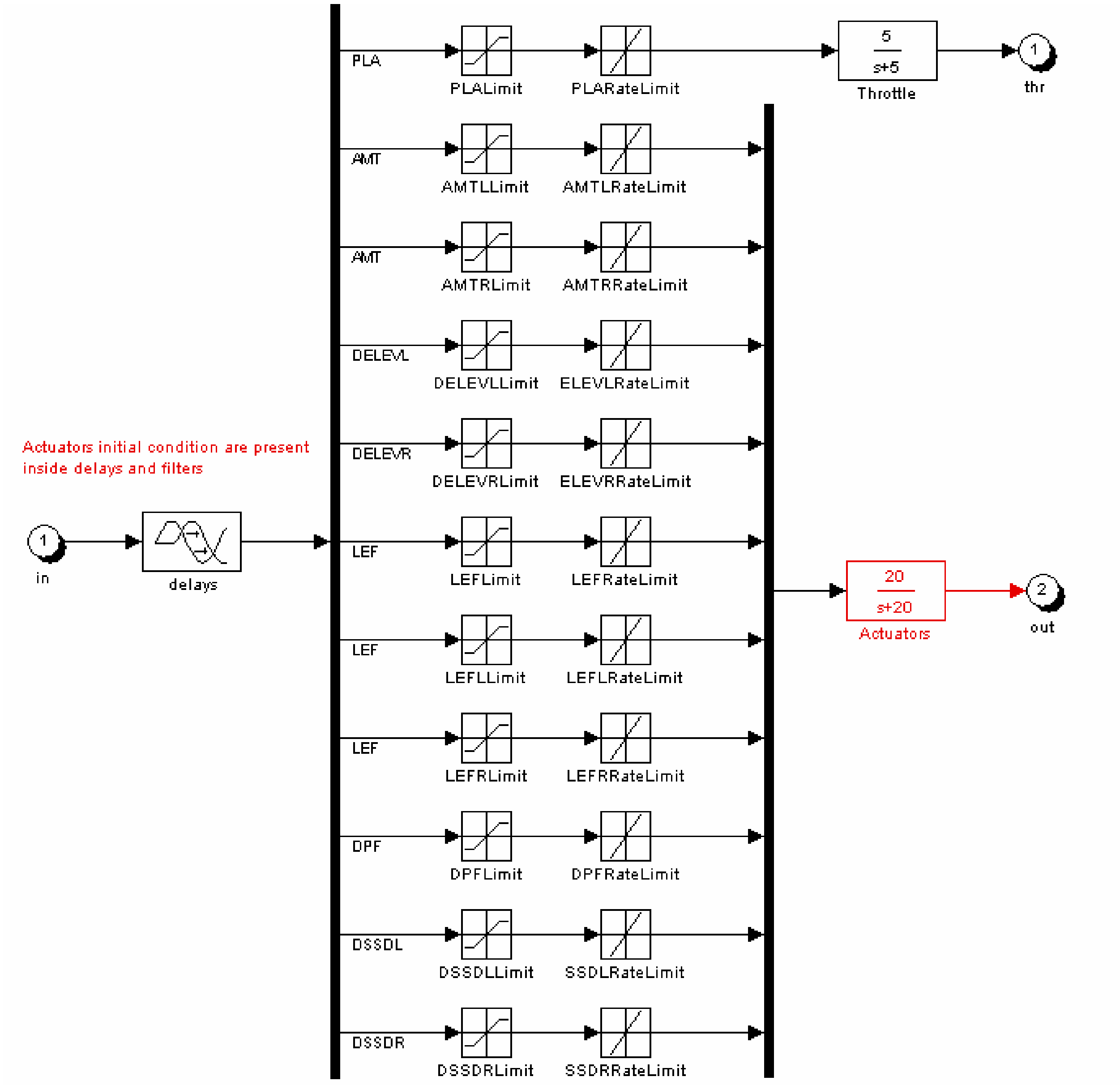


Figure 3.12: Simulink model of the UAV actuator dynamics

Chapter 4 Experimental Setup2

4.1 UAV Software

The UAV software is the vital part of the entire AAR simulation and is explained in detail in this chapter. Figure 4.1 shows the subsystem within the UAV Software block. The major blocks within the UAV software subsystem are the MV block, the Switch and Fusion block, and the UAV controller.

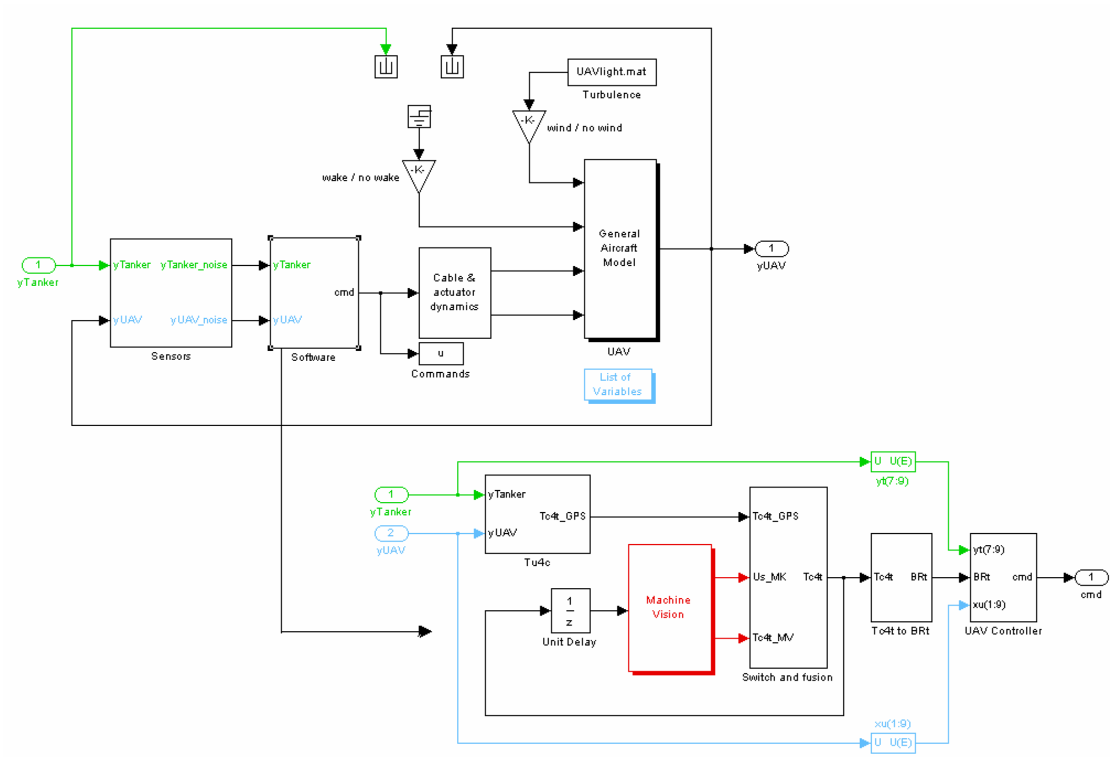


Figure 4.1: Simulink model of the UAV software system and its subsystem

4.1.1 Machine Vision System

The MV system is the block where the actual image processing is performed to detect the corners of the tanker and estimate the distances between the camera and the tanker, based on the corner detection results. The performance of the MV system depends on the performance of the image capturing, corner detection, labeling, and pose estimation algorithms. The SUSAN and Harris algorithms are the corner detection methods, being considered in this effort.

Figure 4.2 presents the MV block from the AAR simulation.

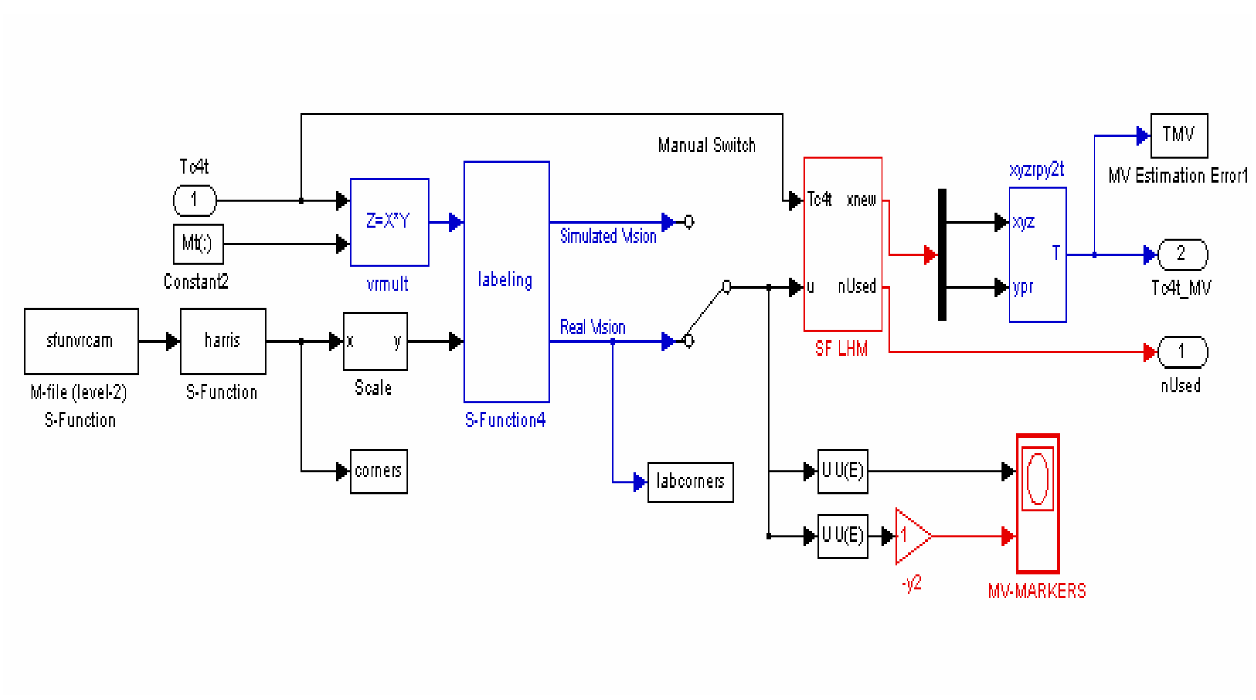


Figure 4.2: Simulink model of the MV system

The MV has one input port (${}^{C}T_{T}$ - input port 1 named Tc4t in the Figure 4.2) -that is- the homogeneous transformation matrix (4 x 4) from TRF to CRF, which is composed as follows:

$$
{}^{C}T_{T} = \left[\begin{array}{ccc|c} & {}^{C}R_{T} & & {}^{T}TC \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{4.1}
$$

where ${}^{C}R_{T}$ is the rotation matrix that changes the reference frame from TRF to CRF, and ${}^{T}TC$ is the translation vector (x, y, z) that originates in the tanker center of gravity

(CG_{TK})and ends in the camera origin. The matrix ${}^{C}T_{T}$ is used to transform a vector expressed within TRF to a corresponding vector in CRF. This matrix contains all the information necessary to express the relationship between the TRF and the CRF. The MV block has two outputs, the first one (*nUsed*) is the number of used corners in the pose estimation problem, and the second one (${}^{C}T_{T}$) is the homogeneous transformation matrix (Equation 4.1), and is provided by the pose estimation algorithm. The subsequent sections deals with explaining each part of the MV system.

4.1.1.1 Image Capture

The camera is a MATALB level-2 M file that captures the image from the Virtual Reality window with a viewpoint from the UAV camera. The image has dimensions (320 x 400). After its capture the image is mapped into the memory as a matrix (320x400x3) where the third dimension represents the red, blue, and green planes. The M-file provides the flexibility to choose from red, green, blue or a gray level image as an output. Figure 4.3 shows a captured image.

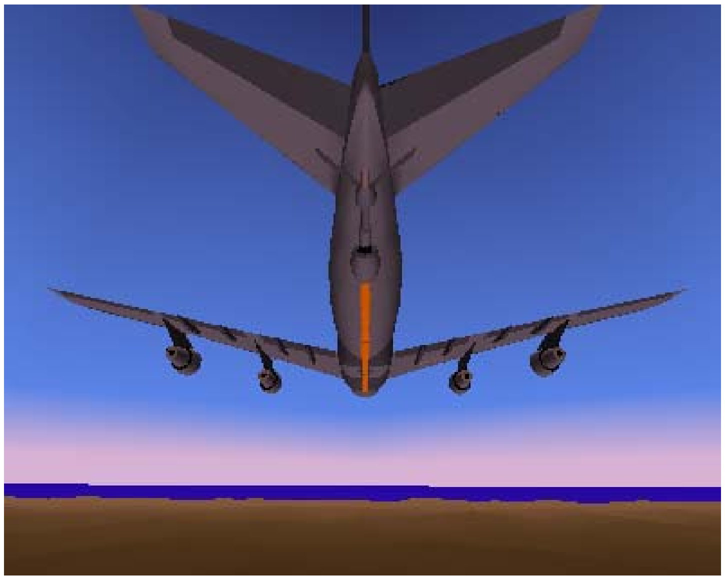


Figure 4.3: Image captured from the VRT

4.1.1.2 Corner Detection

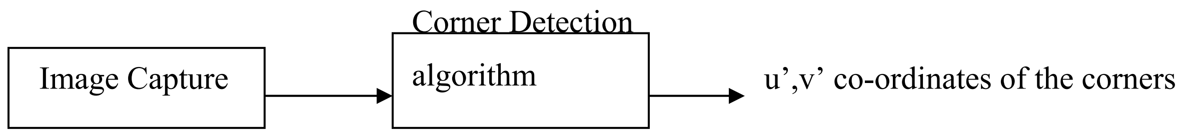


Figure 4.4: Schematic diagram of the corner detection algorithm block

The corner detection/ Feature extraction block is a Simulink level-2 S-Function block. The input to this block is the scaled image captured by the image capture block. The outputs of the corner detection block are the coordinates (u'_j, v'_j) of the corners detected by the algorithm in the camera plane.

4.1.1.3 Scale

The scale function transforms the 2D coordinates of the detected corners from camera plane (u'_j, v'_j), expressed in pixels, into CRF (u_j, v_j), expressed in meters, with the knowledge of the vertical and horizontal dimensions of one pixel and the dimension of the screen

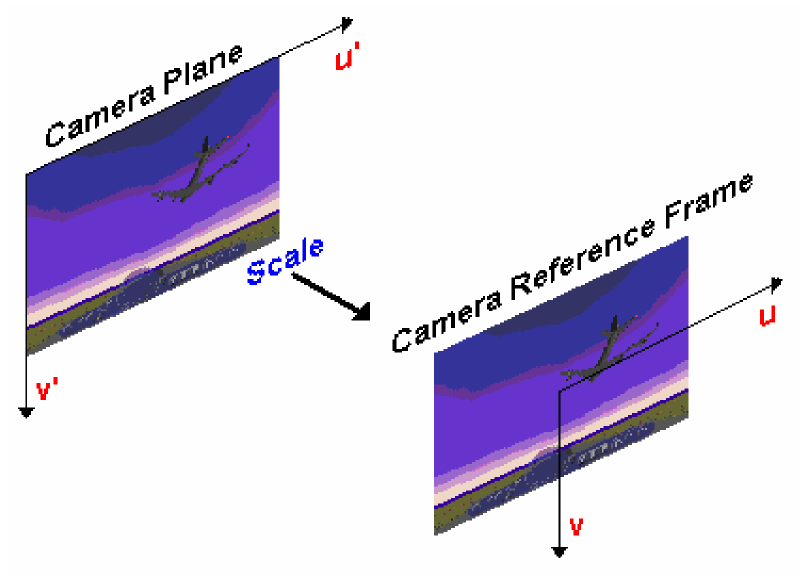


Figure 4.5: The scaling function

4.1.1.4 Physical Corners Transformation

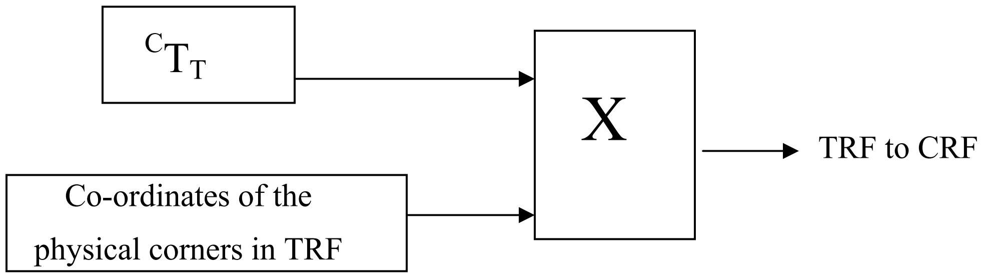


Figure 4.6: Schematic diagram of the physical corners transformation

The MV system assumes that the positions of the physical corners within the TRF (3D coordinates) are known. These coordinates are transformed from the TRF to the CRF using the input to the MV block, which is the transformation matrix $^{C}T_{T}$.

4.1.1.5 Projection Equations

The transformed 3D coordinates of the physical corners are projected onto the camera plane using the "pin-hole" model. The subset $[\hat{u}_j, \hat{v}_j]$ is simply a projection of the corners $P_{(j)}$ in the camera plane [16,58]. Specifically, according to the "pin-hole" model, given a corner '*j*' with coordinates $^{C}P_{(j)} = [^{C}x_j, {^{C}y_j}, {^{C}z_j}, 1]^T$ in the CRF, its projection into the image plane can be calculated using the projection equation:

$$\begin{bmatrix} \hat{u}_j \\ \hat{v}_j \end{bmatrix} = \frac{f}{^{C}x_{p,j}} \begin{bmatrix} ^{C}y_{p,j} \\ ^{C}z_{p,j} \end{bmatrix} = g\left(f, {^{C}T_T(X)} \cdot {^{T}P_{(j)}}\right) \tag{4.2}$$

where *f* is the camera focal length, $^{T}P_{(j)}$ are the components of the corner $P_{(j)}$ in TRF, which are fixed and known 'a priori', and $^{C}T_{T}(X)$ is the transformation matrix between camera and tanker reference frames, which is a function of the current position and orientation vector *X*:

$$X = [^{C}x_T, {^{C}y_T}, {^{C}z_T}, {^{C}\psi_T}, {^{C}\theta_T}, {^{C}\varphi_T}]^T \tag{4.3}$$

For labeling purposes the vector X is assumed to be known. The distance and orientation of the camera in UAV body frame is assumed to be constant and known.

4.1.1.6 Labeling

Once the 2D coordinates of the detected corners on the image plane are found, the problem is to correctly associate each detected corner with its physical feature/corner on the tanker aircraft, whose position in TRF (3D coordinates) is assumed to be known. The general approach is to identify a set of detected corners $[u_j, v_j]$ to be matched to a subset of estimated corner positions $[\hat{u}_j, \hat{v}_j]$. An ad-hoc labeling algorithm that solves the matching problem using a heuristic procedure [3] is implemented in the current work. The outputs of the labeling algorithm are coordinates of the labeled detected corners. Figure 4.5 presents the functioning of the matching and the labeling algorithm. The plots used in Figure 4.5 were simulation outputs for the projected corners, detected corners and the labeled corners.

The 'Points Matching' problem

Once the "projections" subset $[\hat{u}_j, \hat{v}_j]$ is available, the problem of relating the points extracted from the camera measurements to the actual features on the tanker can be formalized in terms of matching the set of points $\{p_1, p_2, ..., p_m\}$ - where $p_j = [u_j, v_j]$ is the generic 'to be matched' point from the camera - to the set of points $\{\hat{p}_1, \hat{p}_2, ..., \hat{p}_n\}$, where $\hat{p}_j = [\hat{u}_j, \hat{v}_j]$ is the generic point obtaining by projecting the known nominal corners in the camera plane through Equation (4.2). Since the two data sets represents the 2D projections of the same points at the same time instant on the same plane, a high degree of correlation between the two sets is expected. In the ideal case, corresponding points would be exactly superimposed, resulting in a trivial matching process.

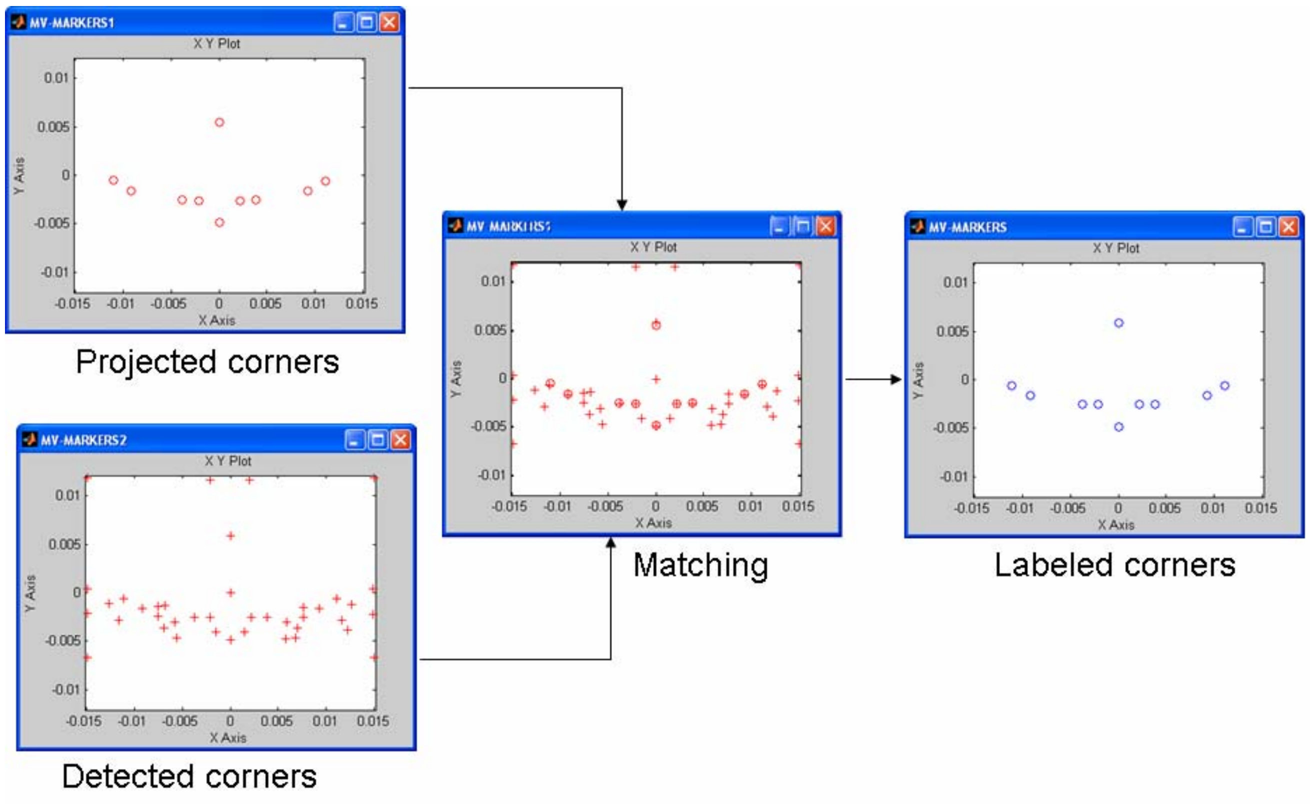


Figure 4.7: The matching and labeling algorithm

4.1.1.7 Simulated Vision and Real Vision

In the MV diagram Figure 4.8, there is a switch that allows the user to choose between "Simulated Vision" (SV) and "Real Vision" (RV) mode. The position of this switch determines which data are provided to the pose estimation algorithm. In the SV mode the pose estimation will be made with the 2D projection of the corners obtained with the "pin-hole" camera model. The data in SV mode are smooth, regular and are complete in each corner position. The SV mode was created to test the performance of the pose estimation algorithm within the MV system for the simulated input. The SV mode allows the use of MV from great distances.

The RV mode allows the use of data from the corner detection and labeling functions, which are likely to be noisy and often incomplete due to some undetectable corners. The camera, the corner detection algorithm, and the labeling algorithm play a major role in the completeness of the data. The RV mode allows the MV system to work with data similar to real-world data, which is the main reason for which this mode has been developed.

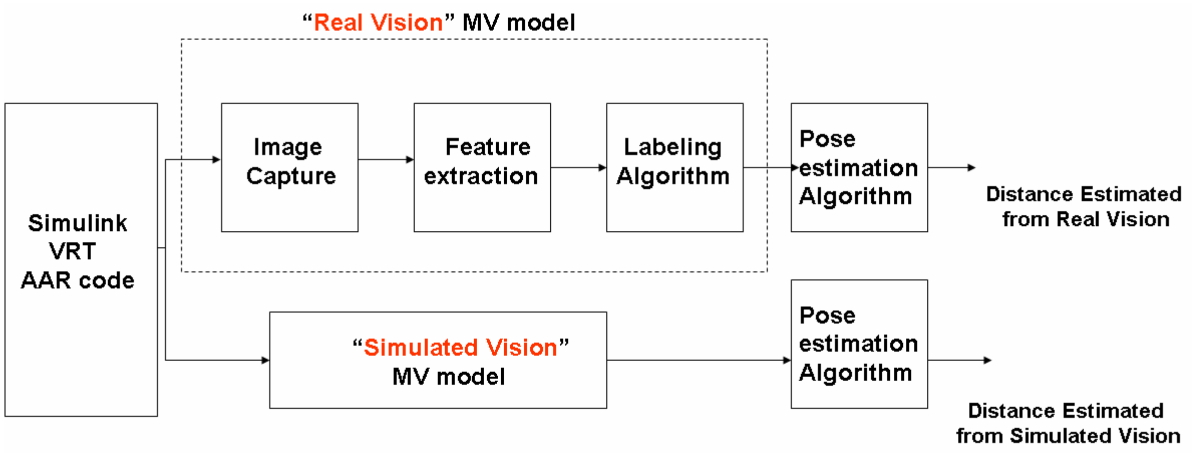

"Real Vision" MV model
Simulink VRT AAR code
Image Capture
Feature extraction
Labeling Algorithm
Pose estimation Algorithm
Distance Estimated from Real Vision
"Simulated Vision" MV model
Pose estimation Algorithm
Distance Estimated from Simulated Vision

Figure 4.8: Block Diagram of the Simulated Vision mode and Real Vision mode

4.1.1.8 Pose Estimation Algorithm

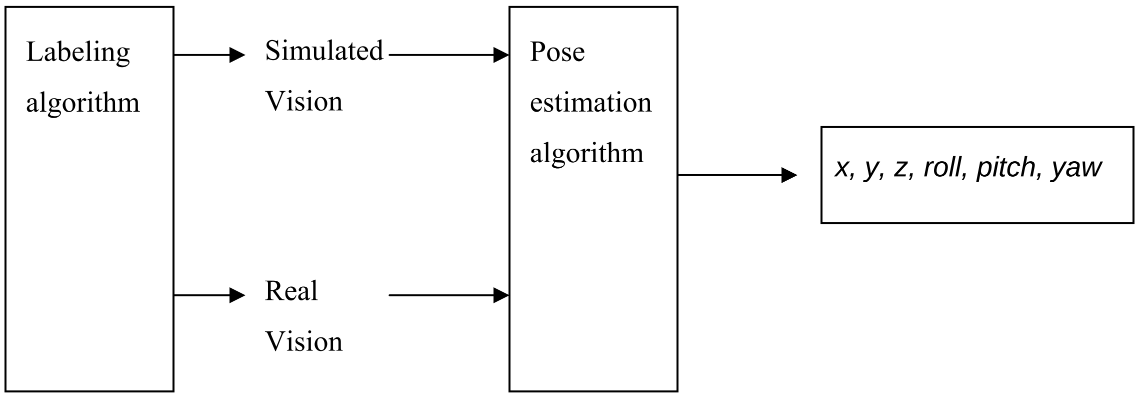
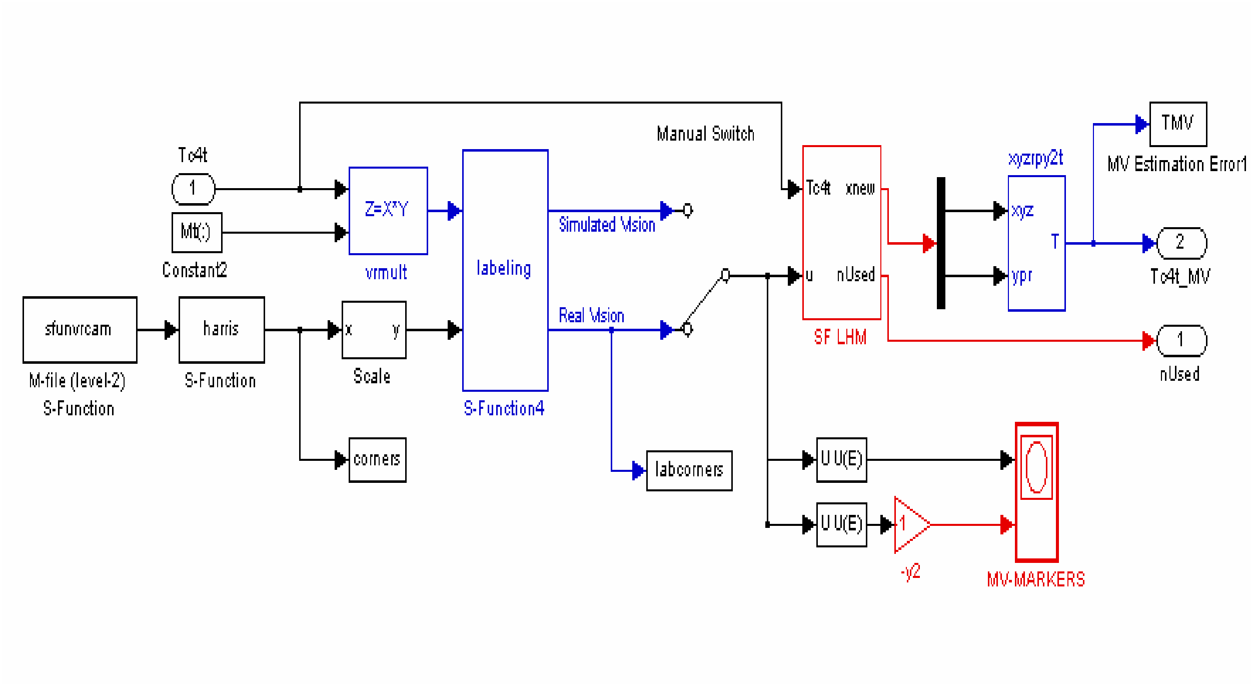


Figure 4.9: Schematic diagram of the pose estimation algorithm block

The information in the set of labeled detected corners is used to derive the rigid transformation relating CRF to TRF using a pose estimation algorithm. Within this study the LHM pose estimation algorithm was used. The LHM algorithms [62] calculate the pose estimation minimizing an error metric based on collinearity in object space. This algorithm is iterative and computes the orthogonal rotation matrix. The iterative calculations without a fixed number of steps can be computationally more intensive. The LHM algorithm provides high robustness, and demonstrated global convergence. The output of a pose estimation algorithm is the vector (*x, y, z, yaw, pitch, roll, nUsed*) that represents respectively the translation vector between TRF and CRF, the relative Euler angles between TRF and CRF and finally the number of corners used to provide the pose estimation. An estimation of the homogeneous transformation matrix $^{C}T_{T}$ is obtained when the pose estimation outputs are processed by the "*xyzrpy2t*" block in Figure 4.2, (reproduced below).



4.1.2 Switch and Fusion

The fusion of the GPS and MV systems is done to ensure a smooth transition from the GPS based data to the MV based data. The basic idea is that the measurement is entirely provided by the GPS system (d_{GPS}) when the UAV is at a distance d from the tanker greater or equal to d_1, while it is entirely provided by the MV system (d_{MV}) if the distance UAV - tanker is lesser or equal to d_2. If $d_1 < d < d_2$ we have a linear interpolation between the distance provided by the MV system and that provided by the GPS system, with the rule:

$$d_F = d_{GPS}\left(1-\frac{d-d_1}{d_2-d_1}\right)+d_{MV}\left(1-\frac{d-d_2}{d_1-d_2}\right) \tag{4.4}$$

The fusion system is shown in Fig 4.10.

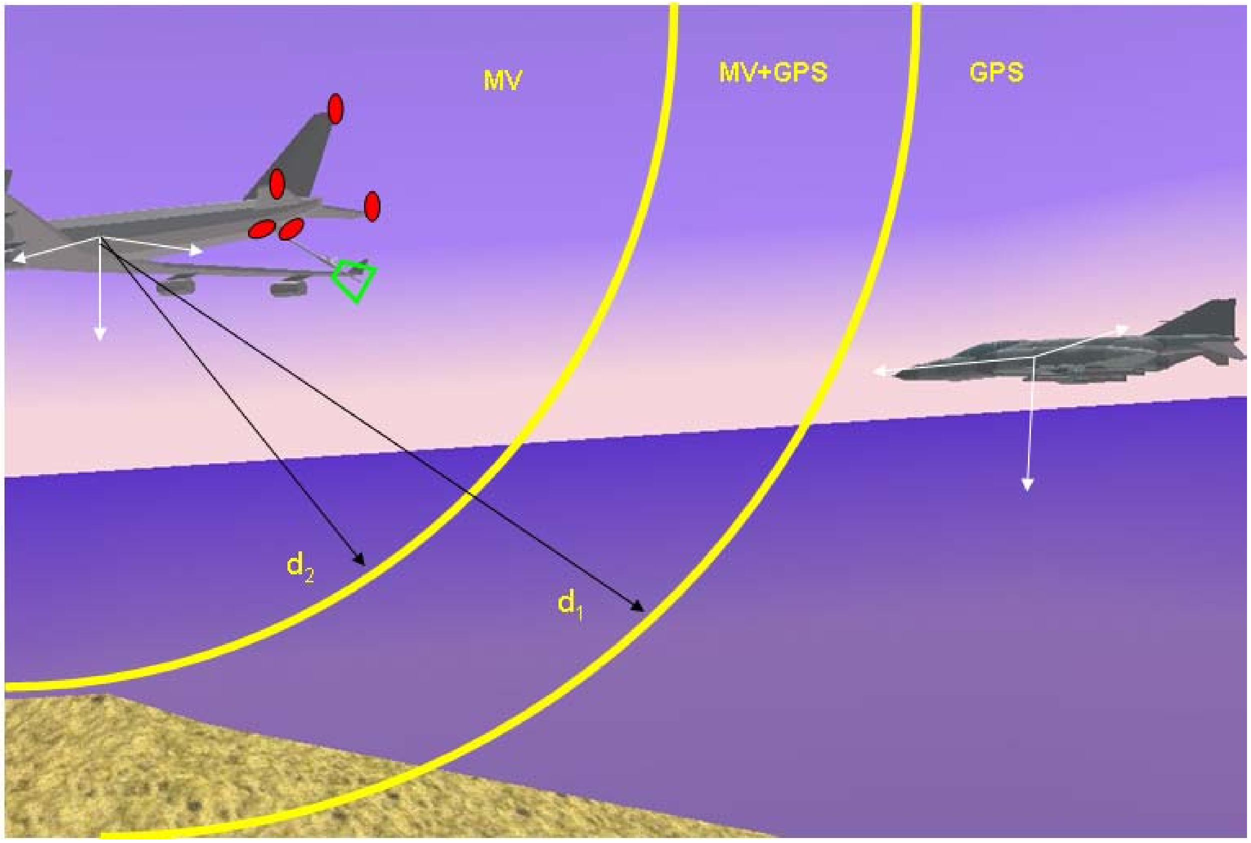


Figure 4.10: GPS and MV system fusion

In more detail, the data provided by the GPS and INU is used to calculate the homogeneous transformation matrix ${}^{C}T_{T}$, which is the matrix that transforms a vector with origin in TRF into a vector with the origin in CRF. ${}^{C}T_{T}$ is also provided by the MV system, and the MV system measurement is valid only if the current measurement is provided with at least 5 corners. Therefore, the fusion is performed if the MV system has atleast 5 or more corners labeled. The fusion is performed only for the translation vector (x, y, z) of the matrix ${}^{C}T_{T}$. The Euler angles provided by the MV system tend to have a larger level of noise than the ones provided by the GPS system. The output of the fusion block is fed back as input to the MV block because this is deemed to be the "best estimation" for the value of ${}^{C}T_{T}$ at the next sampling time.

4.1.3 Controller

The task of the controller is to be able to maintain the aircraft on a defined trajectory, to preserve the internal stability, to follow a docking path, and finally to minimize the distance between receptacle point (R) on the UAV and box point (B) on the tanker. Linear Quadratic Regulator (LQR) approach was used to minimize the distance problem. The LQR control tries to minimize a performance cost function *J* that depends quadratically on the output vector *Y* and the input *U*.

The augmented state vector within this problem is defined as:

$$X_{AUG} = \left[V, \alpha, \beta, p, q, r, \psi, \theta, \varphi, e_x, e_y, e_z, \int e_x, \int e_y, \int e_z \right] \tag{4.5}$$

where e_x, e_y and e_z are the *x, y* and *z* distance between the point R, the point B and the reference trajectory. The performance cost function is defined as:

$$J = \int_0^{\infty} \left(Y^T Q Y + U^T R U \right) dt \tag{4.6}$$

where Q and R are diagonal matrices that establish the performance for each used variable. The output vector *Y* is defined as:

$$Y = \left[e_x, e_y, e_z, \int e_x, \int e_y, \int e_z \right] \tag{4.7}$$

The integral of the distances within the augmented state vector X_{AUG}, guarantees the convergence of the distance to zero. The controller uses the 9 state variables of the UAV system, the vector ${}^{T}BR$ expressed in TRF, and the pitch and roll angles of the tanker (Figure 4.11) to generate the controller command, a vector of 11 elements.

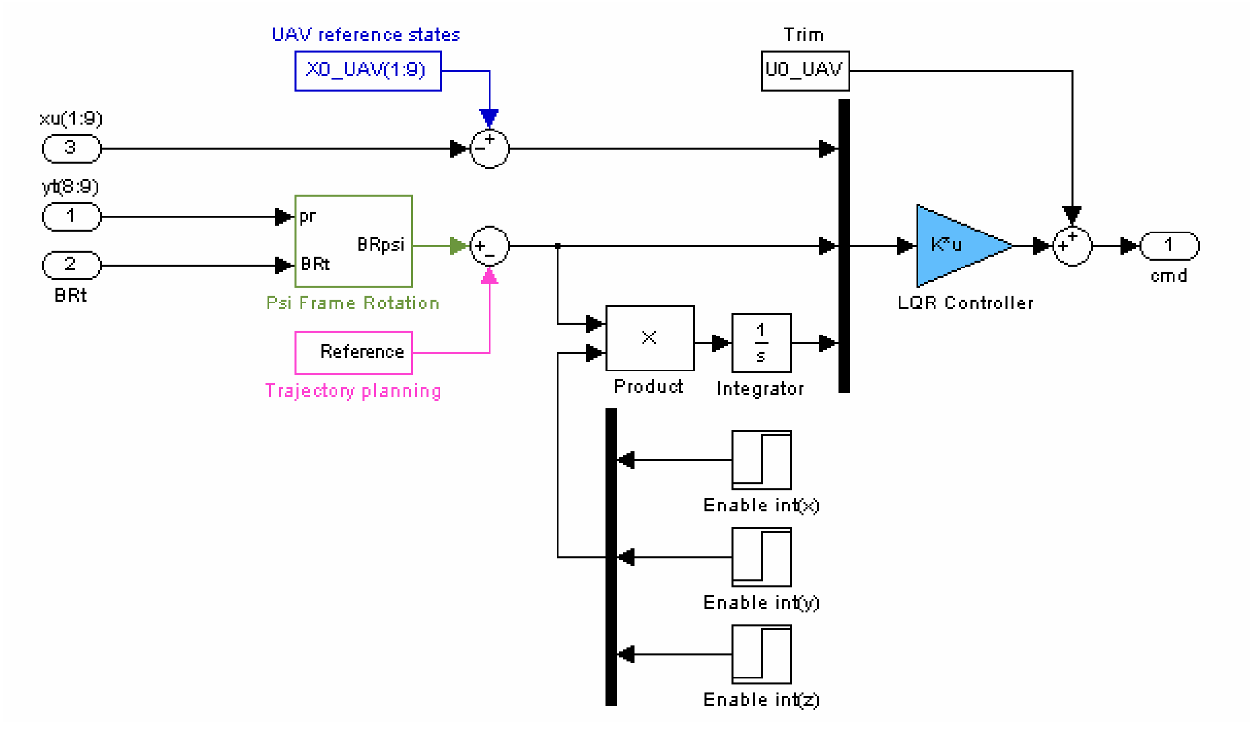


Figure 4.11: Simulink model of the UAV controller scheme

Chapter 5 Experimental Results and Discussions

The experimental results were analyzed and discussed in this chapter. The SUSAN and the Harris corner detector's were the algorithms considered in this work. The two corner detector algorithms were compared initially on a set of four Virtual Reality images of the tanker. The initial test was conducted to obtain the parameter space for the two algorithms. The results obtained from the initial tests were used to refine the parameter space of the two corner detector algorithms for the AAR Simulink simulation. Finally the performances of the two corner detector algorithms were compared on the basis of the AAR Simulink simulation.

5.1 Initial Results

The initial tests were performed to understand the response of the corner detector to a set of Virtual Reality images and to formulate the parameter space for each corner detector. The set of images were obtained from the AAR Simulink simulation at different time intervals, as the UAV approaches the tanker. The simulation was executed in the SV mode to obtain these images. Figure 5.1 shows the set of 3D images used for the initial test. The initial tests were conducted on the 2D grayscale images; Matlab inbuilt command '*rgb2gray*'was used to obtain the grayscale images. A set of corners were defined manually for each image, which represents the true physical corners of the tanker, and were called the physical corners within this approach. Figure 5.2 shows the physical corners for each test image in red.

The performance of the corner detector is dependent on the accuracy with which the physical corners are detected. The confidence measures were defined as *Detection rate* and the *False alarm rate* for this initial study [59]. These rates, which quantify the results, are based on:

- TP (true positives): Detected points that correspond to the set of physical corners.
- FP (false positives): Detected points that do not correspond to the set of physical corners

- FN (false negatives): Undetected physical corners.

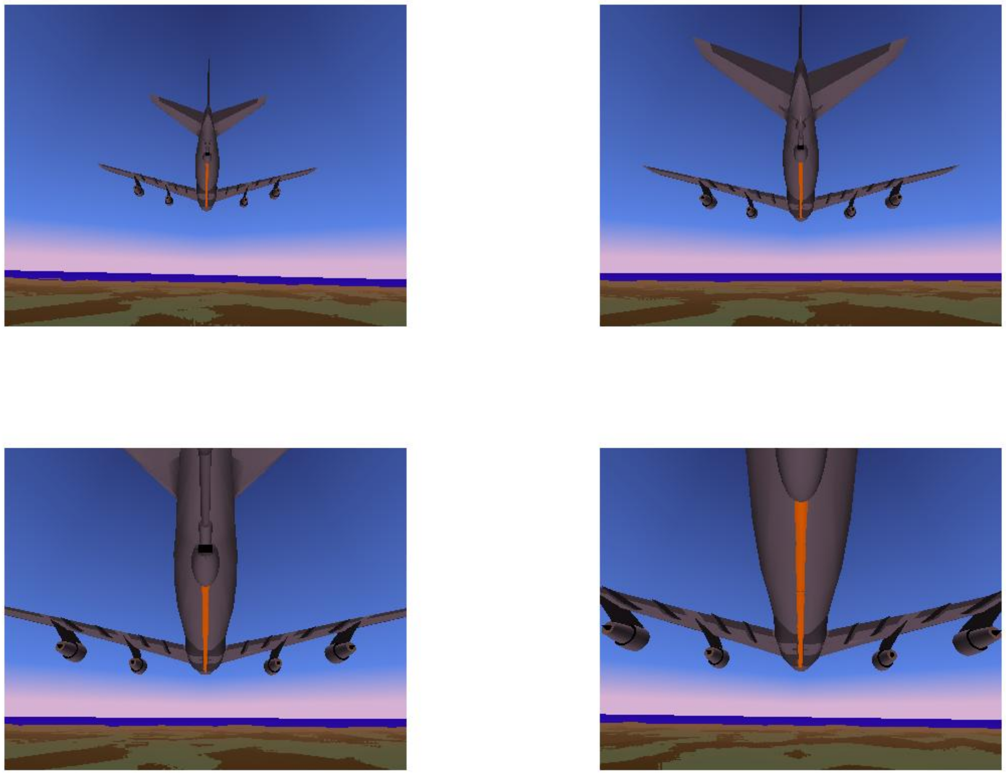


Figure 5.1: Test images obtained from VRT

Based on the above scalars, the *Detection rate* and the *False alarm rate* are defined as:

$$DR = \frac{TP}{TP + FN} \qquad FAR = \frac{FP}{TP + FP} \tag{5.1}$$

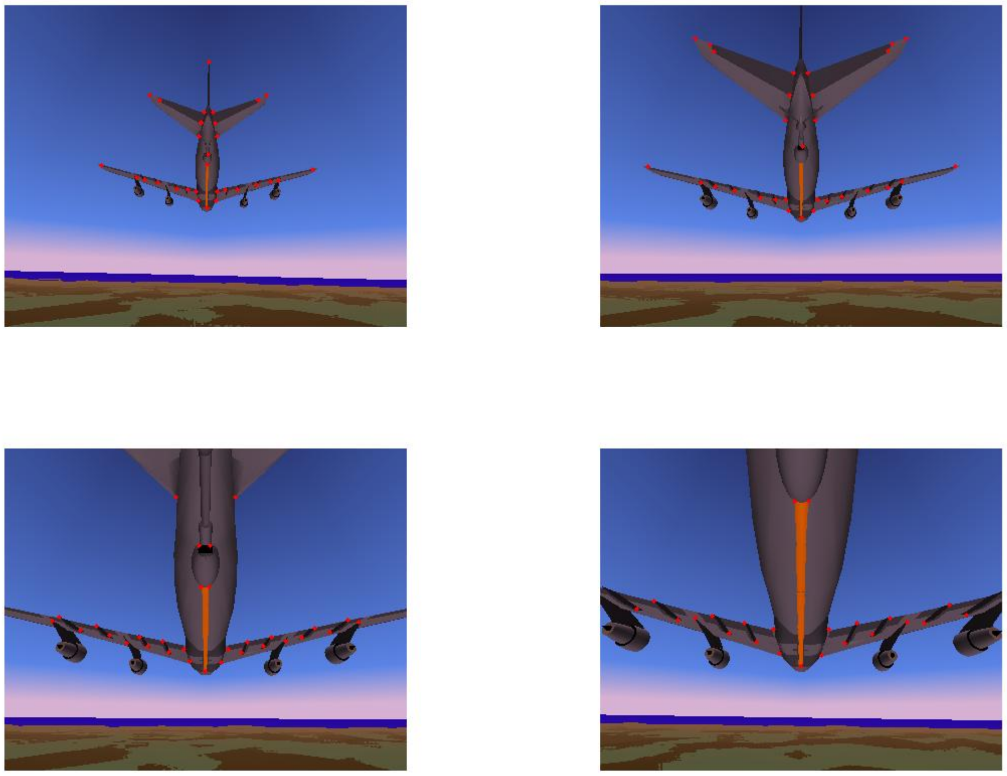


Figure 5.2: Test images with the physical corners marked

5.1.1 Harris Corner Detector

The parameters which affect the performance of the Harris corner detector are the size of the Gaussian mask used to perform the smoothing, the size of the non-maximal suppression mask and the threshold value used to threshold the cornerness map. The initial sets of values for each parameter were obtained on a trail and error basis and were used to refine the parameter sets for the later studies. The considered values for each parameter are shown below:

- *Sigma* – The standard deviance of the Gaussian mask – (2, 3, 4, 5, 6)
- *Non maximal suppression mask size* – (3x3, 5x5, and 7x7 mask size)
- *Threshold* value – (200, 400, 800, 1000, 1200, 1400)

Table 5.1 shows the initial test results for the parameter *sigma*. The tests were conducted with the *threshold* value fixed at a value of 400 and with a 3x3 *non-maximal suppression mask size*.

Sigma	**2**	**3**	**4**	**5**	**6**
Image1					
DR	0.9062	0.7812	0.5937	0.2812	0.1875
FAR	0.6506	0.5	0.4062	0.4705	0.53846
Image2					
DR	0.9032	0.9354	0.8387	0.4838	0.2581
FAR	0.7941	0.5977	0.4583	0.5161	0.6363
Image3					
DR	0.9355	0.9032	0.7419	0.6129	0.3871
FAR	0.84492	0.7171	0.5576	0.5476	0.6129
Image4					
DR	0.9565	1	0.9565	0.7391	0.6087
FAR	0.9052	0.7946	0.6811	0.65306	0.6666

Table 5.1: Initial study results for *sigma* parameter

The optimum parameter value would have high detection rate and low false alarm rate. From Table 5.1 it is obvious that good detection rates were provided by the *sigma* values 2 and 3, and decent detection rates by the *sigma* values 4 and 5, specifically for the last two images, which identify with the docking phase. Though the *sigma* value of 2 displayed high detection rate, the false alarm rate was also pretty high when compared to the other values. Hence the values 3, 4 and 5 were chosen to provide the parameter space for parameter *sigma.*

Table 5.2 shows the initial study results for the *threshold* parameter. The tests were conducted with the *sigma* value set to 3 and with a 3x3 *non-maximal suppression mask size*.

Threshold	**200**	**400**	**600**	**800**	**1000**	**1200**	**1400**
Image1							
DR	0.78125	0.78125	0.71875	0.5	0.4687	0.4375	0.4375
FAR	0.7524	0.6753	0.6290	0.6923	0.6739	0.6744	0.6666
Image2							
DR	0.9354	0.87097	0.6129	0.5483	0.5483	0.5483	0.5483
FAR	0.7289	0.6666	0.6885	0.6666	0.5853	0.5526	0.5404
Image3							
DR	0.9032	0.9032	0.9032	0.8387	0.8387	0.8387	0.6774
FAR	0.8082	0.7544	0.6956	0.6790	0.6285	0.6232	0.65
Image4							
DR	1	1	0.9130	0.9130	0.8261	0.73913	0.6521
FAR	0.8244	0.7745	0.7586	0.7307	0.6984	0.6964	0.6938

Table 5.2: Initial study results for *threshold* parameter

The *threshold* values within 200-800 provided good detection rate and false alarm rate and hence were chosen to form the parameter space of the *threshold* parameter.

Table 5.3 shows the initial study results for the non-maximum suppression mask size. The tests were conducted with the *sigma* value set to 3 and the *threshold* value set to 400.

Non-maximal suppression mask size	**3x3**	**5x5**	**7x7**
Image1			
DR	0.78125	0.8333	0.8333
FAR	0.6478	0.6428	0.6153
Image2			
DR	0.67742	0.67742	0.67742
FAR	0.7	0.6818	0.6379
Image3			
DR	0.90323	0.96552	0.89655
FAR	0.7254	0.6923	0.6708
Image4			
DR	1	1	1
FAR	0.7909	0.7767	0.7578

Table 5.3: Initial study results for *non-maximal suppression mask size* parameter

All the considered *non-maximal suppression mask sizes* provided similar detection rate and false alarm rates. The size of the mask is directly proportional to the execution time; thus the bigger the mask the longer is the execution time. Hence taking into consideration the time factor, the 3x3 and 5x5 mask sizes were chosen to form the parameter space for the *non-maximal suppression mask size*.

Hence the final values of each parameter, as a result of the initial study were:

Sigma: [3, 4, 5]

Threshold: 200-800

Non-maximal suppression mask sizes: 3x3, 5x5.

5.1.2 SUSAN Corner Detector

The parameters which affect the performance of the SUSAN corner detector are the *mask size* used to calculate the USAN area, the size of the *non-maximal suppression* *mask*, and the *brightness threshold* value. The initial sets of values for each parameter were obtained on a trail and error basis and were used to refine the parameter set for the later studies.

- *Mask sizes* : masks with 9 , 37 and 57 pixels
- *Non-maximal suppression mask sizes*: 5x5, 7x7, 9x9 and 11x11 mask sizes
- *Brightness threshold*: [20, 28, 36]

Table 5.4 shows the initial test results for the *mask size* parameter. The tests were conducted with the *brightness threshold* value set to 28 and with a 7x7 sized *non-maximal suppression mask size.*

Mask size	**9 pixels mask**	**37 pixels mask**	**57 pixels mask**
Image1			
DR	0.75	0.8125	0.4062
FAR	0.6307	0.5438	0.315
Image2			
DR	0.6128	0.7742	0.322
FAR	0.7764	0.6923	0.696
Image3			
DR	0.7419	0.8709	0.4516
FAR	0.8203	0.7244	0.674
Image4			
DR	0.7826	0.8261	0.4782
FAR	0.871	0.8288	0.7555

Table 5.4: Initial study results for the *mask size* parameter

Based on the detection rates the 9 pixels mask and the 37 pixels mask were chosen to form the parameter space for the *mask size* parameter.

Table 5.5 shows the initial study results for the *non-maximal suppression mask size* parameter. The tests were conducted with the 37 pixels *mask size* and with the *brightness threshold* value set to 28.

Non-maximal suppression mask size	**5x5**	**7x7**	**9x9**	**11x11**
Image1				
DR	0.8125	0.8125	0.8125	0.7187
FAR	0.63889	0.5438	0.409	0.3235
Image2				
DR	0.774	0.7742	0.7742	0.7742
FAR	0.752	0.6923	0.6065	0.5
Image3				
DR	0.8709	0.8709	0.8709	0.8064
FAR	0.7731	0.7244	0.6747	0.6527
Image4				
DR	0.82609	0.82609	0.82609	0.7826
FAR	0.8416	0.8224	0.7865	0.7464

Table 5.5: Initial study results for the non-*maximal suppression mask size* parameter

The 5x5, 7x7 and 9x9 suppression mask sizes provided similar detection rates, and hence were chosen to form the parameter space for the *non-maximal suppression mask sizes*.

Table 5.6 shows the initial study results for the *brightness threshold* parameter. The tests were conducted with the 37 pixels *mask size* and with 9x9 for *non-maximal suppression mask size.*

Brightness threshold	**20**	**28**	**36**
Image1			
DR	0.8437	0.8125	0.3
FAR	0.66	0.50	0.7
Image2			
DR	0.8709	0.7742	0.4839
FAR	0.7127	0.6417	0.6808
Image3			
DR	0.9677	0.9310	0.9275
FAR	0.7368	0.689	0.6571
Image4			
DR	0.9565	0.82609	0.82609
FAR	0.8382	0.7957	0.724

Table 5.6: Initial study results for the *brightness threshold* parameter

The detection rates were similar for all the considered cases especially during the docking phase, and hence were chosen to form the parameter space of the *brightness threshold* parameter.

Hence the final values of each parameter, as a result of the initial study were:

Mask size: 9 pixels and the 37 pixels masks

Non-maximal suppression mask size: 5x5, 7x7, 9x9

Brightness threshold: [20, 28, 36]

As mentioned earlier the initial study results were basically used to obtain the range of values for each parameter affecting the corner detectors.

5.2 ROC Curves

The SUSAN and the Harris corner detector algorithms were also compared based on the ROC curves. Receiving Operating Characteristic (ROC) curve is a graphical plot of the sensitivity Vs specificity for a system as its discrimination threshold is varied. Bowyer et al proposed methods for using the ROC techniques to evaluate the performance of edge detectors [60] and Martinez-Fonte et al proposed the usage of the ROC curves to evaluate the performance of the Harris and SUSAN corner detector [61]. The ROC curves are given by the *% undetected corners* versus the *% false alarm rate*. *% false alarm rate* would be the ratio of the number of false positives to the number of pixels in the image, and the *% undetected corners* would be the ratio of the number of false negatives to the number of physical corners specified manually [60,61]. The false positives are given by the number of points detected which do not correspond to a true corner and the false negatives are given by the number of undetected corners.

For a given image and a detector, the ROC curve is obtained by sampling the parameter space, and producing a set of points based on the best false positives and false negatives. If the detector has *n* parameter each with $x_1, x_2, x_3 ....x_n$ values, then the number of points in the parameter space would be the product of $x_1, x_2, x_3 ....x_n$. Each image is analyzed with each point from the parameter space and *false positives* and the *false negatives* are calculated. Each calculated *false positives* and *false negatives* plot a point in the ROC space, but only those values close to the origin will be included in the ROC curve [60,61].

For the Harris corner detector, 5 values were considered for the *sigma* parameter, 5 values for the *threshold* parameter and the 3 mask sizes for the *non-maximal suppression*, totaling to 75 points in the parameter space. Each of the four images was analyzed with each point from the parameter space and points for the ROC curves were obtained. For the SUSAN corner detector, 6 values were considered for the *brightness threshold* value, 3 mask sizes and 4 non-maximum suppression masks, totaling to 72 points in the parameter space. Each of the four images was analyzed with each point from the parameter space and points for the ROC curves were obtained. Figure 5.3 shows the ROC curves for each image using both the corner detectors.

The curves to the left in the ROC plot indicate a better performing corner detector, since the false negatives are less compared to the curves on the right. The ROC curves for the second and the fourth image showed better performance by the Harris corner detector, while the ROC curves for the first and third images showed better performance by the SUSAN corner detector. Based on these results it was expected that both the corner detector algorithms would exhibit similar performances when executed within the AAR simulation.

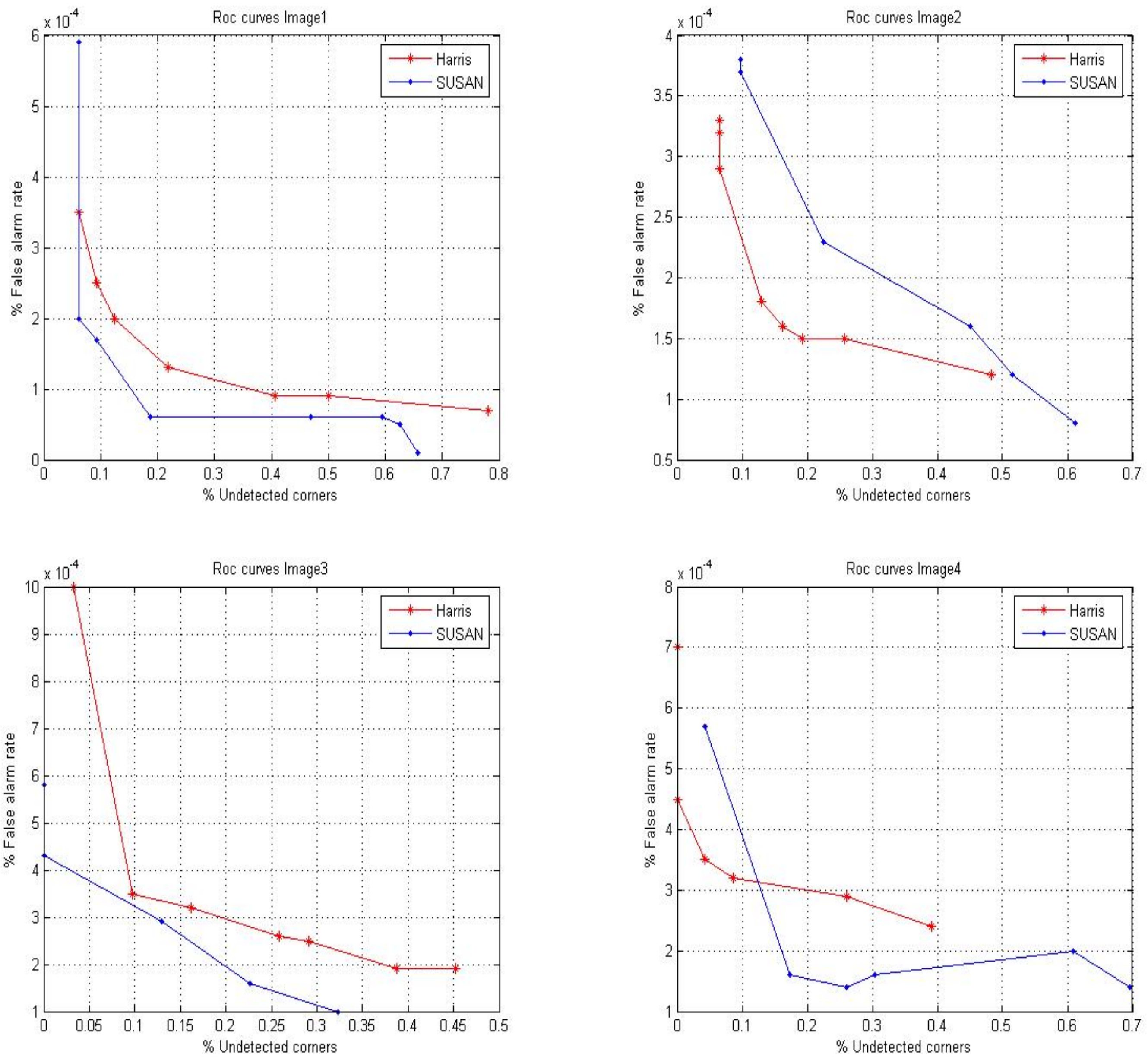


Figure .5.3: ROC curves comparing the SUSAN and Harris corner detector

5.3 Parameter Setup

This study was conducted to finalize the parameter values for the Harris and SUSAN corner detectors when implemented within the closed loop AAR simulation as Simulink level 2 S-Functions. The algorithms were compared based on the MV estimation error, the number of detected corners, and the distance between the 3D window and the receptacle, the complete run time of the simulation and the maximum number of false alarms. The MV estimation is the linear position estimation obtained by the pose estimation algorithm in the ‘x’, ‘y’ and ‘z’ directions. These values were compared with reference values, which are the readings from the simulated sensors, and the difference is termed as the MV estimation error. The total estimation error is given by the equation:

$$e = \sqrt{x^2 + y^2 + z^2} \tag{5.2}$$

where *x, y* and *z* are the MV estimation errors in the x, y and the z directions. ${}^{E}BR$, is the distance between the receptacle and the boom expressed in ERF. The ‘x’, ‘y’ and ‘z’ coordinates of the ${}^{E}BR$ were compared with the reference values, which are the center coordinates of the 3D window and the difference is termed as ${}^{E}BR$ error.

The AAR Simulink simulation was executed in the SV mode to select a set of physical corners repeatedly detected, especially during the docking phase. These corners were defined as the physical corners for the AAR simulation. Fig 5.4 shows the physical corners so selected.

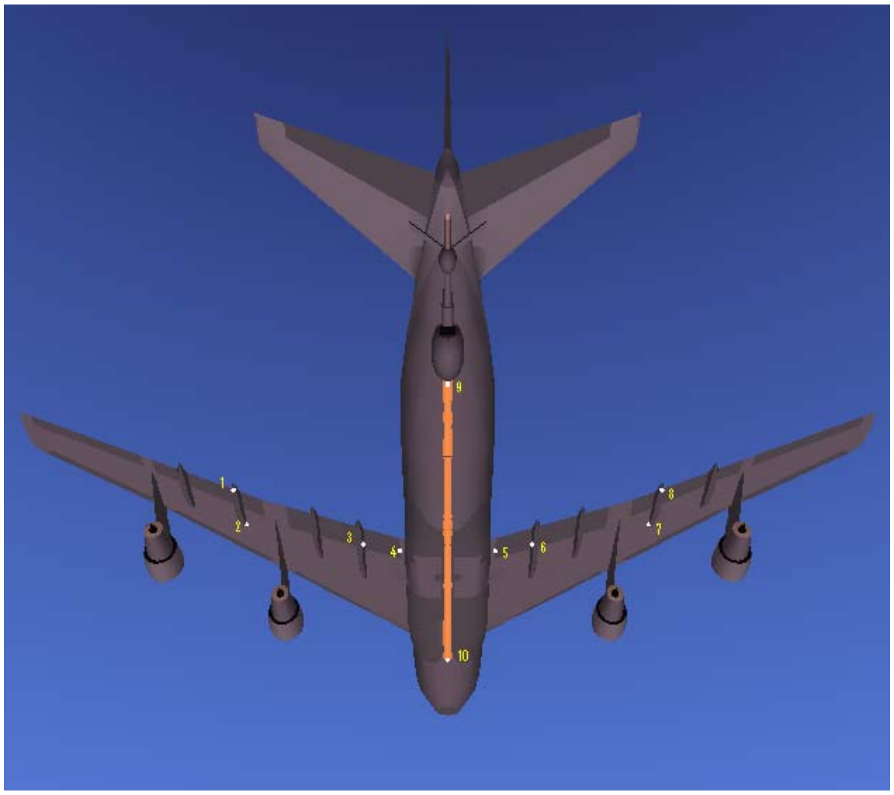


Figure 5.4: The physical corners selected for the AAR simulation are marked and numbered in white

5.3.1 Harris Corner Detector Parameters

5.3.1.1 Sigma parameter

Figure 5.5 shows the AAR simulation results with *sigma* values from the initial study results. The first subplot shows the total estimation error, the second subplot is the total estimation error during the last 40 sec of the AAR simulation, and the third subplot shows the number of detected corners.

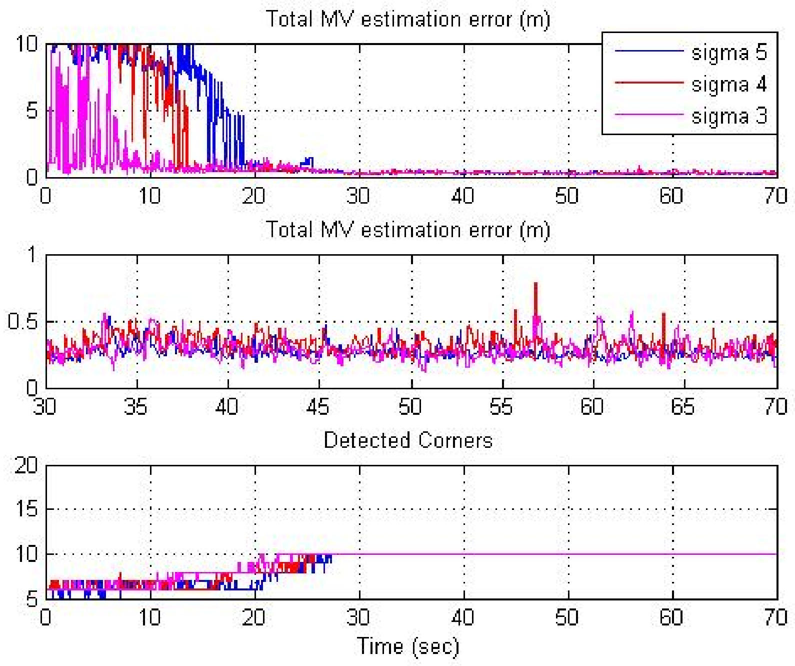


Figure 5.5: AAR Simulation results for *sigma* parameter

Sigma	MV rmsx	MV rmxy	MV rmsz	BRe rmsx	BRe rmsy	BRe rmsz	Time (sec)	%Avg DC	Max (FA)
5	0.0647	0.0400	0.1592	0.1751	0.0816	0.1599	354	100	53
4	0.0902	0.0522	0.1782	0.1492	0.0835	0.1647	372	100	77
3	0.0665	0.0603	0.1658	0.1721	0.0881	0.1674	370	100	102

Table 5.7: Summary of the AAR simulation results for *sigma* parameter

Table 5.7 summarizes the AAR simulation results for different values of *sigma* during the last 30 sec of the simulation. The MV rmsx, MV rmsy and the MV rmsz are the root mean squares of the MV estimation errors in the ‘x’, ‘y’ and ‘z’ directions. BRe rmsx, BRe rmsy, and BRe rmsz are the root mean squares of ^{E}BR errors in the ‘x’, ‘y’ and ‘z’ directions. Time specifies the total run time of the AAR simulation. Simulink ‘profiler’ was used to obtain the total run time value. %Avg DC is the percentage of the average number of corners detected during the last 30 sec, and max (FA) is the maximum number of false positives/false alarms detected during the simulation. The results, only during the last 30 seconds of the simulation were analyzed and tabulated emphasizing the importance of the performance of the corner detectors during the docking phase. The *sigma* value of 5, produced the least errors with minimum false alarms. The total run time of the simulation was also less when compared with other *sigma* values.

5.3.1.2 Non-maximal suppression mask size parameter

Figure 5.6 shows the AAR simulation results with *non-maximal suppression mask sizes* from the initial study results. The simulation was executed with the *sigma* value set to 5 and the *threshold* value set to 250. Table 5.8 summarizes the AAR simulation results for different *non-maximal suppression mask sizes* during the last 30 sec of the simulation.

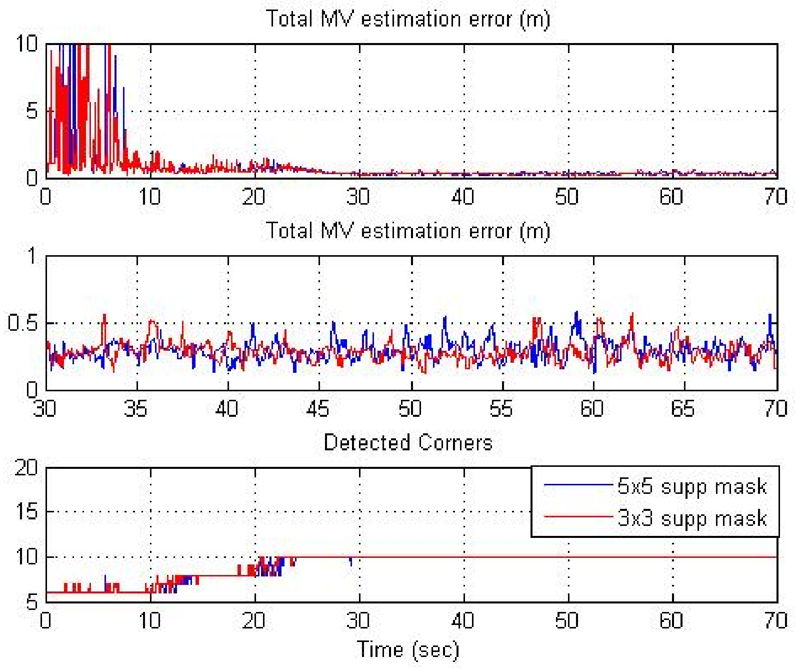


Figure 5.6: AAR simulation results for *non-maximal suppression mask sizes* parameter

	MV rmsx	MV rmxy	MV rmsz	BRe rmsx	BRe rmsy	Bre rmsz	Time (sec)	%Avg DC	Max (FA)
3x3	0.0665	0.0603	0.1658	0.1721	0.0881	0.1674	370	100	102
5x5	0.0803	0.0601	0.1751	0.1732	0.0921	0.1674	400	100	91

Table 5.8: Summary of the AAR simulation results for *non-maximal suppression mask size* parameter

The results were almost similar with both mask sizes, but the run time for the 3x3 mask size was less when compared to the other mask size.

5.3.1.3 Threshold parameter

Figure 5.7 shows the AAR simulation results with *threshold* values from the initial study results.

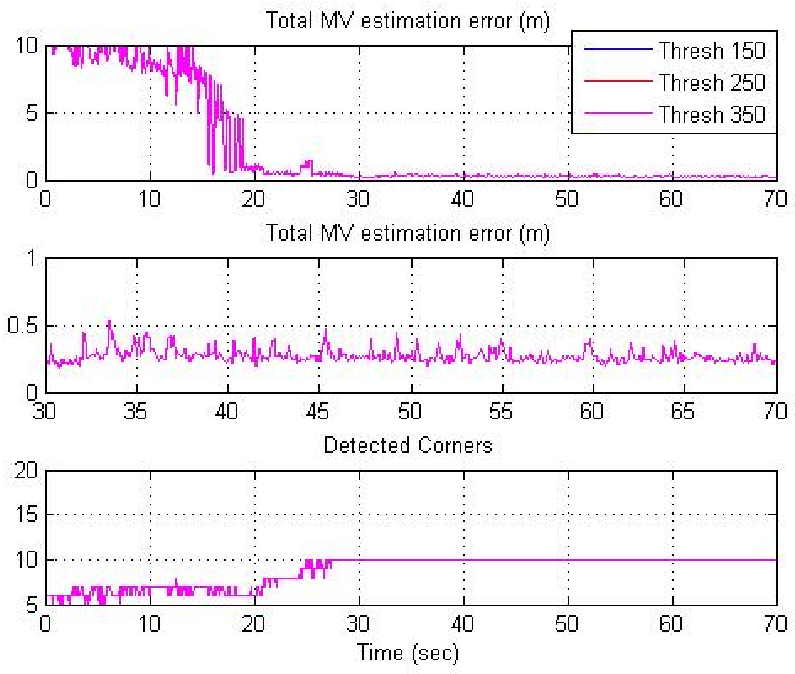


Figure 5.7: AAR Simulation results for *threshold* parameter

The *threshold* values chosen were pretty close to have any significant difference on the AAR simulation results. The *threshold* value of 250 showed better response with a *sigma* value of 5, and hence was selected to be the final value for this parameter.

The finalized parameter values for the successful execution of the closed loop AAR simulation with Harris corner detector were:

Sigma: 5

Non-maximal suppression mask: 3x3

Threshold: 250

5.3.2 SUSAN Corner Detector Parameters

5.3.2.1 Mask size parameter

Figure 5.8 shows the AAR simulation results with *mask sizes* from the initial study results. The AAR simulation was executed with 7x7 *non-maximal suppression mask size* and *brightness threshold* value set to 26. Table 5.9 summarizes the AAR simulation results for different *mask sizes* during the last 30 sec of the simulation.

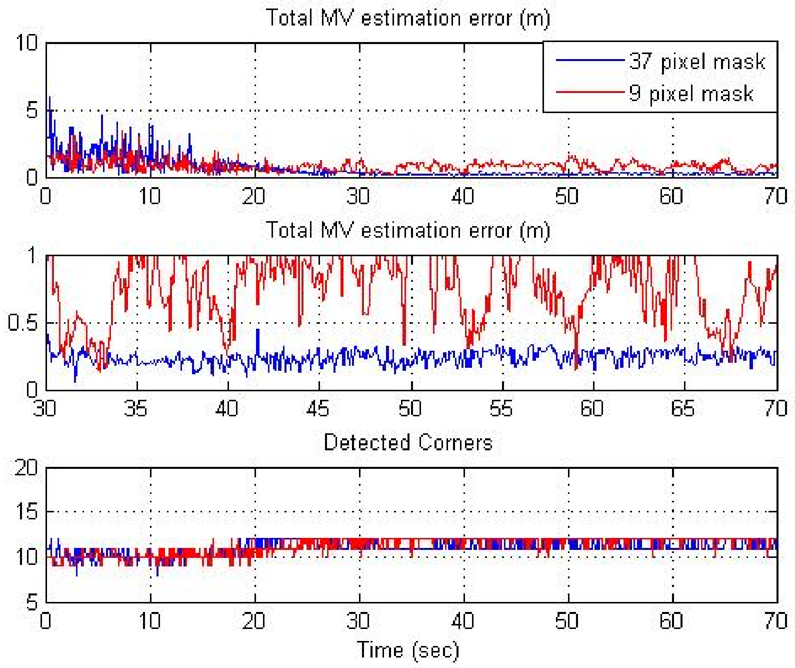


Figure 5.8: AAR simulation results for the *mask size* parameter

	MV rmsx	MV rmsy	MV rmsz	BRe rmsx	BRe rmsy	BRe rmsz	Time (sec)	%Avg DC	Max (FA)
37pix	0.1467	0.0413	0.0569	0.3394	0.0887	0.1579	225	94.606	106
9 pix	0.4194	0.0868	0.0369	0.3534	0.1044	0.1929	230	98.586	147

Table 5.9: Summary of the AAR simulation results for *mask size* parameter

The performance of the corner detector with the 37 pixels mask size exhibited better detection and less error when compared to the performance of the corner detector with the 9 pixels mask size. Even though the percentage of the average detected corners is more for the 9 pixels mask, far too many false positives were detected. Localization of the corners was slightly different for the 9-pixel mask, leading to a larger error when compared with the physical corners.

5.3.2.2 Non-maximal suppression mask size parameter

Figure 5.9 shows the AAR simulation results with *non-maximal suppression mask sizes* from the initial study results. The simulation was executed with the 37 pixels *mask size*, and the *brightness threshold* value set to 26. Table 5.10 summarizes the AAR simulation results for different *non-maximal suppression mask sizes* during the last 30 sec of the simulation.

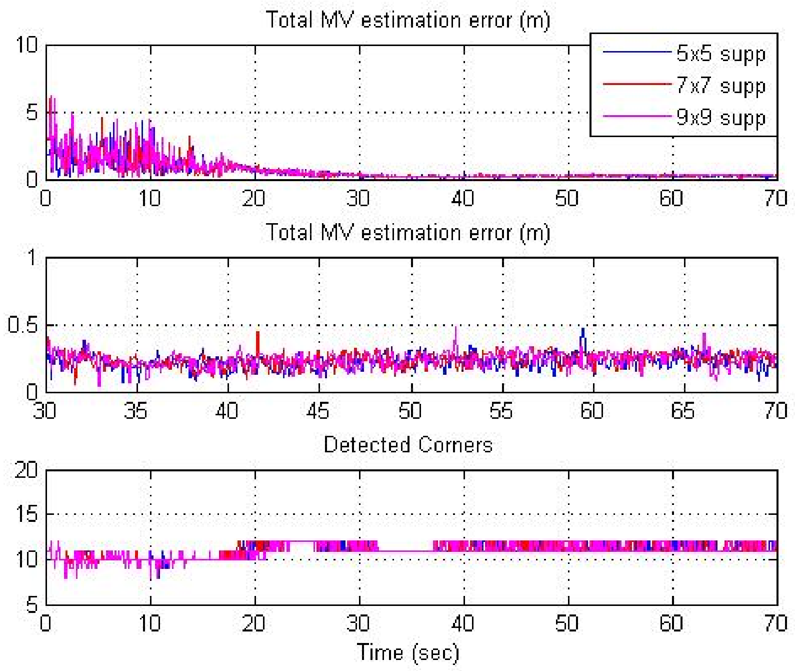


Figure 5.9: AAR simulation results for the *non-maximal suppression mask size* parameter

	MV rmsx	**MV rmsy**	**MV rmsz**	**BRe rmsx**	**BRe rmsy**	**BRe rmsz**	**Time (sec)**	**%Avg DC**	**Max (FA)**
5x5	0.1426	0.0357	0.0523	0.3376	0.0875	0.1517	238	94.66	125
7x7	0.1467	0.0413	0.0569	0.3394	0.0887	0.1579	225	94.606	106
9x9	0.1453	0.0398	0.0611	0.3322	0.0869	0.1638	236	94.523	83

Table 5.10: Summary of the AAR simulation results for the *non-maximal suppression mask size* parameter

The results for all the three suppression mask sizes were similar. Among the three mask sizes the complete simulation run time for the 7x7 mask was the minimum. Too many operations were required by the 5x5 mask, and intensive calculations by the 9x9 mask.

5.3.2.3 Brightness threshold parameter

Figure 5.10 shows the AAR simulation results with *brightness threshold* values from the initial study results. The simulation was executed with 37 pixels *mask size* and 7x7 *non-maximal suppression mask size*. Table 5.11 summarizes the AAR simulation results for different *brightness threshold* values during the last 30 sec of the simulation.

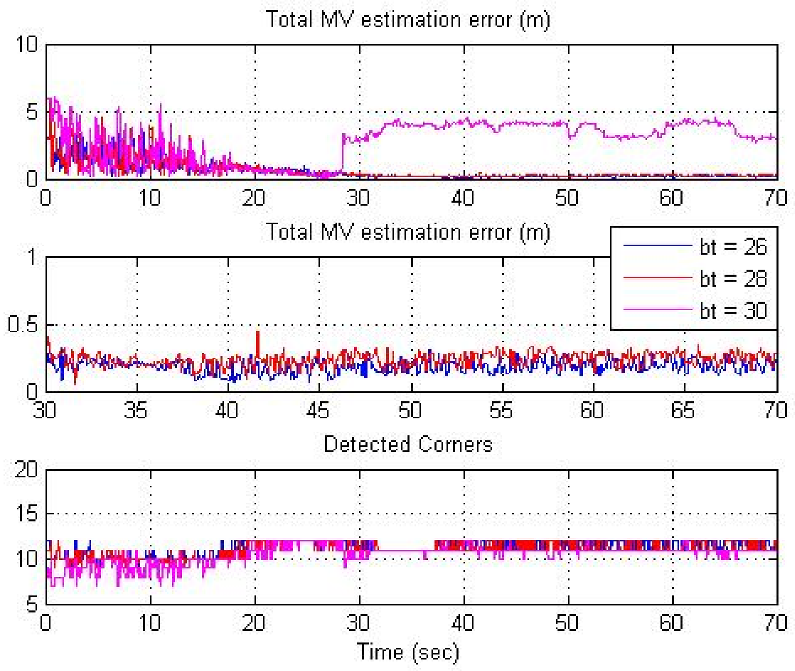


Figure 5.10: AAR simulation results for *brightness threshold* parameter

	MV rmsx	MV rmsy	MV rmsz	BRe rmsx	BRe rmsy	BRe rmsz	Time (sec)	%Avg DC	Max (FA)
26	0.1089	0.04217	0.0414	0.3148	0.0837	0.1202	213	97.241	115
28	0.1467	0.0413	0.0569	0.3394	0.0887	0.1578	225	94.60	106
30	2.328	0.5038	0.6082	2.2151	0.5078	0.5860	224	91.556	95

Table 5.11: Summary of the AAR simulation results for the *brightness threshold* parameter

As already stated in the theory, the larger the value of *brightness threshold*, the lesser the number of corners detected, which is the case with the *brightness threshold* value 30. Most of the physical corners were undetected which led to more errors. The *brightness threshold* value of 26 produced minimum errors and had minimum run time when compared to the other values.

The finalized parameter values for the successful execution of the closed loop AAR simulation with SUSAN corner detector were:

Mask Size: 37 pixels mask

Non-maximal suppression mask size: 7x7 mask

Brightness Threshold: 26

5.4 Comparison of the corner detector algorithms

The following performance criteria were introduced for a detailed comparison of the two algorithms in terms of required computational speed, accuracy and robustness.

5.4.1 Speed Performance

The computational speed of the Harris and SUSAN routines depends in general on the usage of system resources. Within this study the corner detector algorithms were written in "C" and implemented as 'Level 2 Simulink S-Function blocks'. A Pentium 4, 3.20 GHz desktop with 1 GB of RAM was used for this analysis. The speed performance was measured with the Simulink® "profiler" tool, which provides the run time in seconds for each called function and sub-function. On average, the SUSAN and the Harris algorithms require 0.0182 sec and 0.1249 sec respectively for each simulation step. Therefore the SUSAN algorithm is approximately 7 times faster than the Harris algorithm.

5.4.2 Accuracy

The accuracy of the corner detector is based on the MV estimation. The MV estimation is the linear position estimation in the 'x', 'y' and 'z' directions. The closer the MV estimation to the reference values, the better the performance of the corner detector and better the accuracy of the corner detector in terms of corner detection. Figures 5.11 and 5.12 show the MV estimation by the Harris and the SUSAN algorithm. They show the linear position estimation obtained by the pose estimation algorithm when Harris (Figure 5.11) and SUSAN (Figure 5.12) were used as corner detection algorithms. Within these figures, 'x real', 'y real' and 'z real' are the reference values, calculated using the readings from the simulated sensors, whereas 'x MV', 'y MV' and the 'z MV' are the values provided by the pose estimation algorithm.

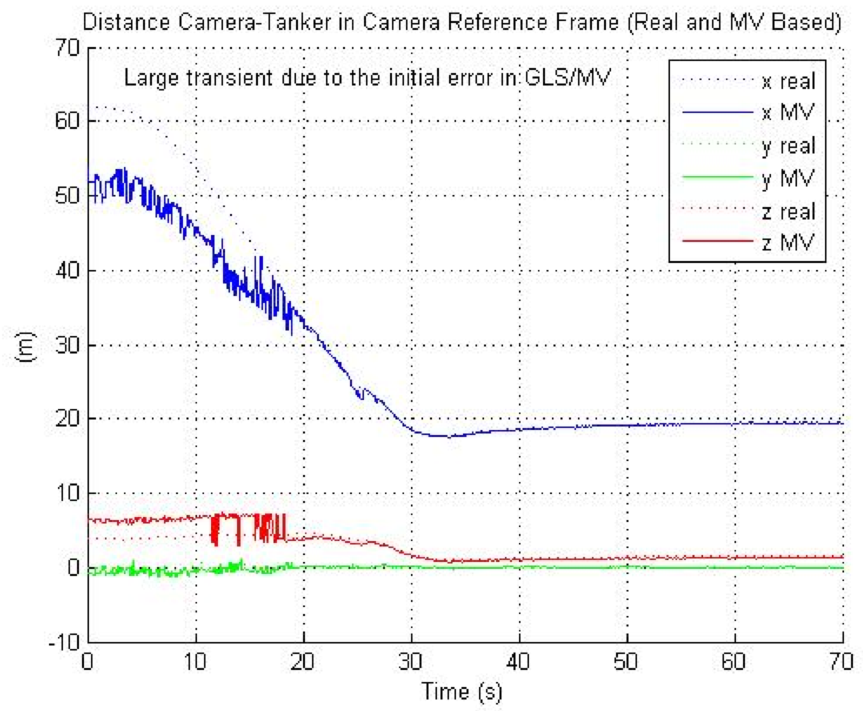


Figure 5.11: 'Real' x,y,z Vs 'Estimated' x,y,z for the Harris corner detector

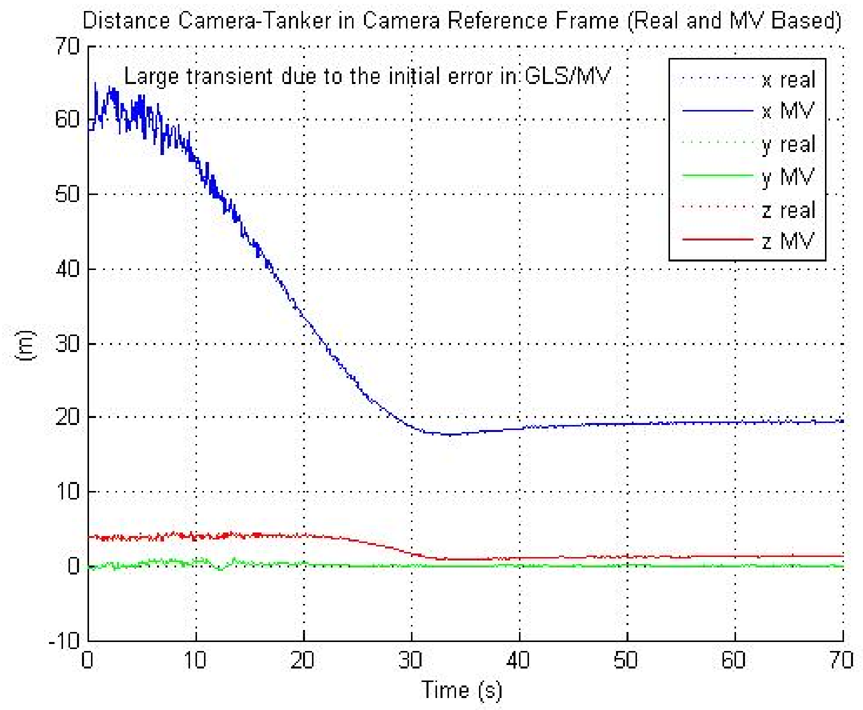


Figure 5.12: 'Real' *x y z* Vs 'Estimated' *x y z* for the SUSAN corner detector

The responses of both the corner detector algorithms were almost similar during the last 40 sec of the simulation, which corresponds with the docking sequence, and also when the MV system comes into action. Total estimation error (Equation.5.2) was also used to compare the two corner detector algorithms. Figure 5.13 shows the total estimation error for the complete simulation run time (first subplot), during the last 40 seconds of the simulation (second subplot) and the number of detected corners (third subplot). The physical corners specified for the SUSAN algorithm were 12, and for the Harris algorithm were 10. The actual percentage of the detected physical corners for each algorithm is shown in Table 5.12.

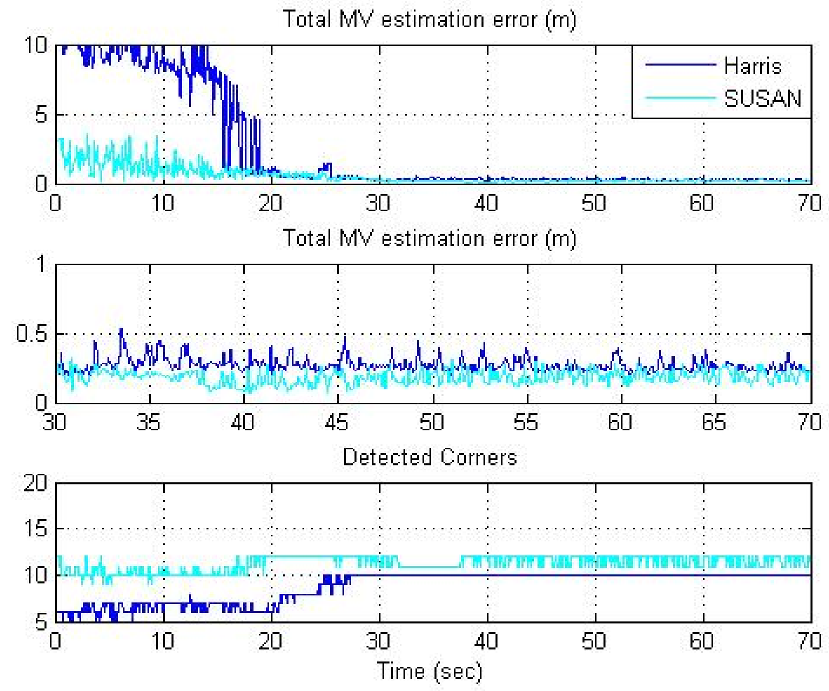


Figure 5.13: Total estimation errors of the Harris and SUSAN corner detectors

	MV rmsx	MV Rmsy	MV rmsz	BRe rmsx	BRe rmsy	BRe rmsz	Time (sec)	%Avg DC	Max (FA)
Harris	0.0647	0.04	0.1593	0.1751	0.0816	0.1599	387	100	53
SUSAN	0.1089	0.0421	0.0414	0.3148	0.0827	0.1203	221	97.24	115

Table 5.12: Comparison of the estimation errors of Harris and the SUSAN corner detectors

The Harris corner detector had a better MV estimation in the 'x' direction when compared to the SUSAN corner detector. The poor performance by the SUSAN corner detector in the 'x' direction, led to a large ${}^{E}BR$ error in the 'x' direction. Though both the corner detectors had almost similar responses during the last 30 sec of the simulation the Harris corner detector had an edge over the SUSAN corner detector regarding the estimation in the 'x' direction.

The accuracy of the corner detectors was also based on the distance between the 3D window, the receptacle, and the reference trajectory. It was observed that both the corner detectors satisfy the desired limits of the 3D window specification (given in Table 3.1) in the 'y' and 'z' directions, and the allowable limit in the 'x' direction, during the docking phase. The distance between the 3D window, the receptacle, and the reference trajectory in the 'x' direction reached the desired limit, with Harris corner detector within the last 20 sec of the simulation, and with the SUSAN corner detector only during the last 5 sec of the simulation.

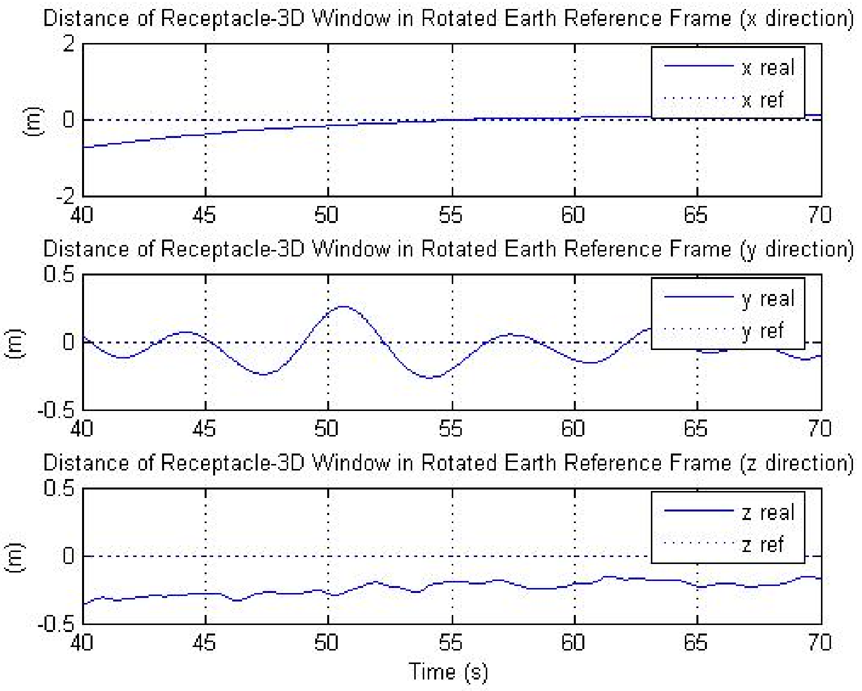


Figure 5.14: Distance of receptacle and 3D window with Harris corner detector

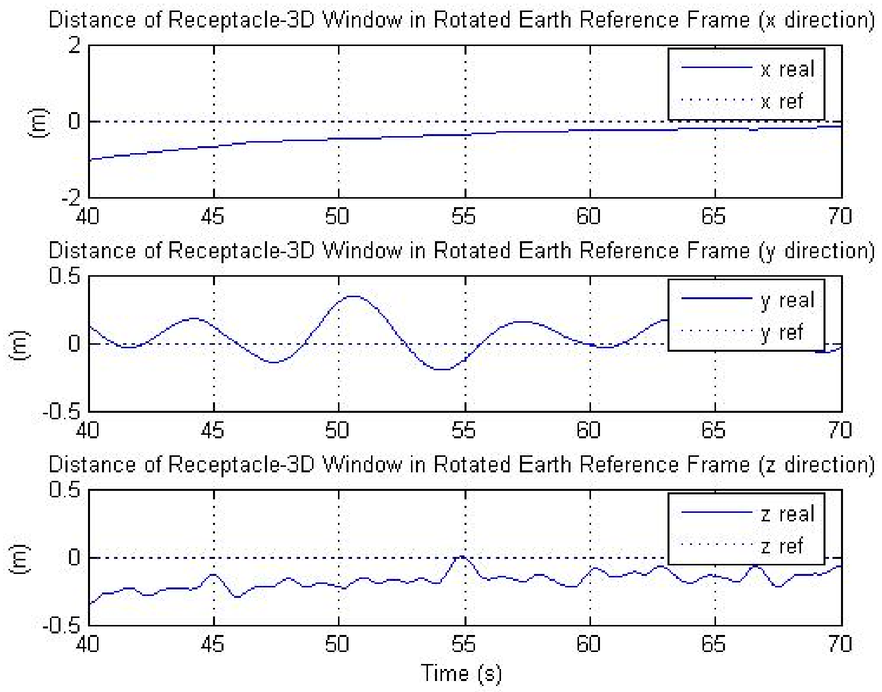


Figure 5.15: Distance of receptacle and 3D window with SUSAN corner detector

5.4.3 Robustness Study

A robustness analysis was conducted by evaluating the performance of the corner detection algorithms in the event of the following image perturbations:

- Presence of noise in the image
- Variations in image contrast.
- Motion Blur

5.4.3.1 Noise addition to the input image

This analysis was performed to evaluate the stability of the corner detection algorithms in the presence of Gaussian white noise in the image.

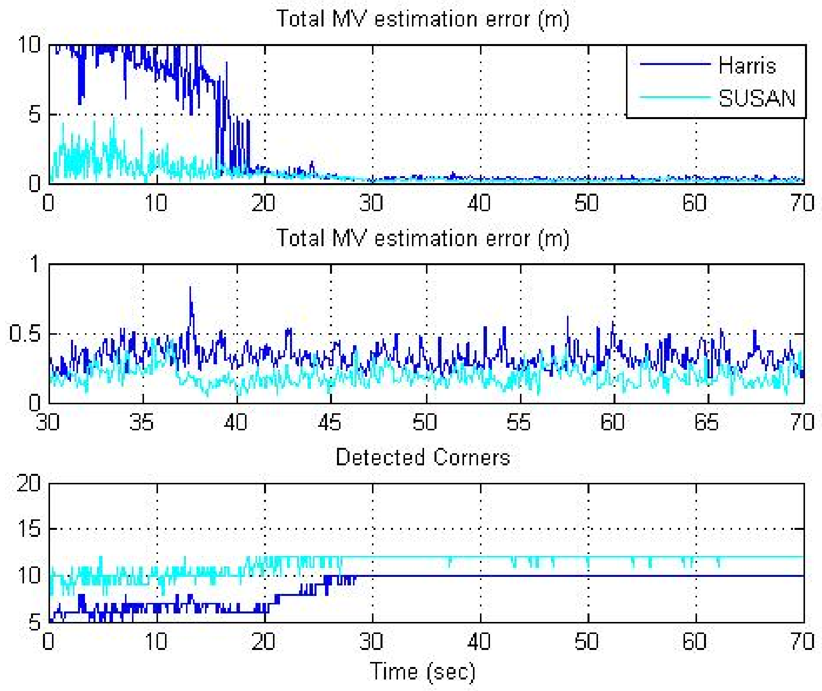


Figure 5.16: Total estimation errors of the Harris and SUSAN corner detector with added Noise

The presence of the noise was simulated using the Matlab command "*imnoise*". A Gaussian noise with zero mean and with 0.001 variance was introduced into the image stream. Figure 5.16 shows the total MV estimation error (first subplot), total MV estimation error during the last 40 sec of the simulation (second subplot), along with the number of detected corners (third subplot), for both algorithms. Both the algorithms respond almost similarly to the added noise.

5.4.3.2 Poor contrast image

This analysis was performed to evaluate the robustness of the corner detection algorithms in the presence of poor contrast conditions. These conditions can be induced by different factors such as the lightning effects from the sun and/or foggy weather resulting in low contrast tanker images.

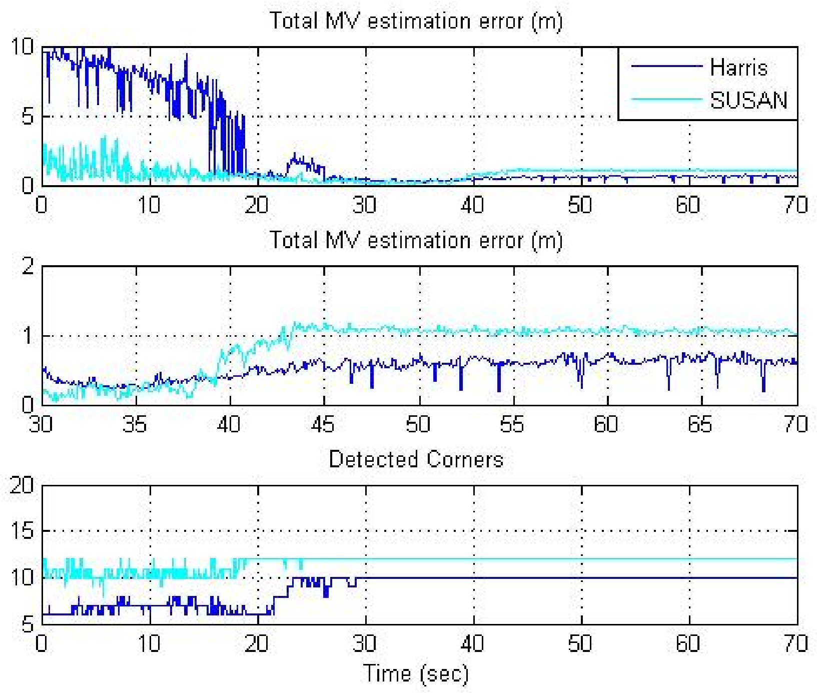


Figure 5.17: Total estimation errors of the Harris and SUSAN corner detector with varied contrast

Figure 5.17 shows the total MV estimation error (first subplot), total MV estimation error during the last 40 sec of the simulation (second subplot), along with the number of detected corners (third subplot), for both algorithms. The contrast of the image was reduced using the Matlab "*imadjust*" command. Both the algorithms showed deterioration in the estimation, but SUSAN corner detector was observed to be more sensitive to the low contrast images. Reducing the *brightness threshold* value to almost 10 from 26 produced the above results, for the SUSAN corner detector.

5.4.3.3 Motion blur

Image blurring occurs when there is a substantial relative movement of the object within the timeframe of the image capture. An analysis was performed to compare the performance of the corner detection algorithms to blurred images. Specifically, the Matlab command '*imfilter*' was used to generate a motion blur of 3 pixels on the image stream. Figure 5.18 shows the total MV estimation errors, along with the number of detected corners under the above conditions. The SUSAN corner detector was more sensitive to the motion blur when compared to the Harris corner detector.

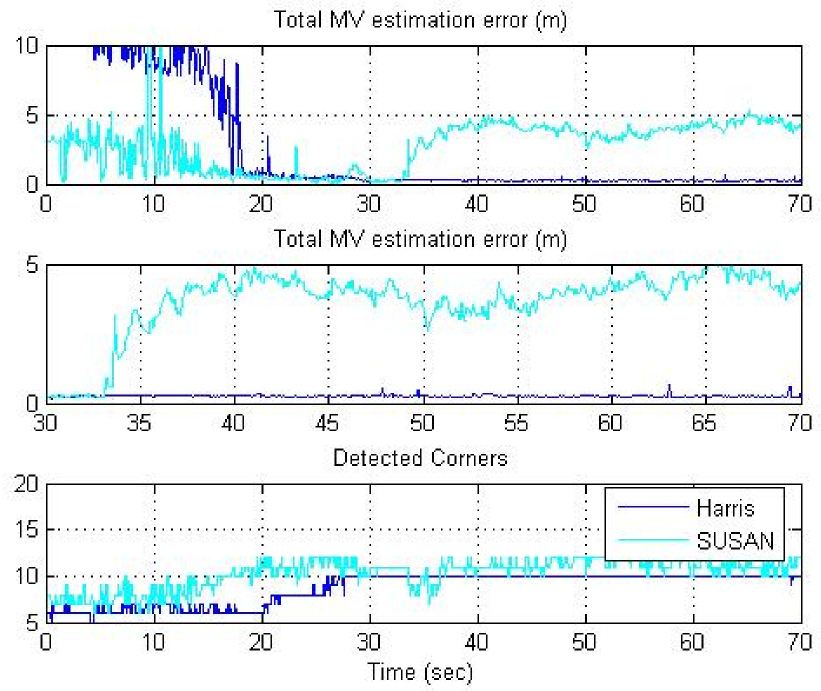


Figure 5.18: Total estimation errors of the Harris and the SUSAN corner detectors with motion blur

Table 5.13 summarizes the results of the robustness study through the root mean square errors of the x, y and z coordinates, of the MV estimation error as well as of the ${}^{E}BR$ error, along with the percentage of the average numbers of corners detected, and the maximum number of false alarms, for both algorithms, under normal and perturbed conditions.

From the robustness study it was evident that the Harris corner detector provided better results and is robust to the changes induced into the images.

	MV rmsx	**MV rmsy**	**MV rmsz**	**BRe rmsx**	**BRe rmsy**	**BRe rmsz**	**%Avg DC**	**Max (FA)**
Harris (Normal)	0.0647	0.04	0.1593	0.1751	0.0816	0.1599	100	53
SUSAN (Normal)	0.1089	0.0421	0.0414	0.3148	0.0827	0.1203	97.24	115
Harris (Noise)	0.1015	0.04924	0.1835	0.1546	0.0787	0.1721	100	127
SUSAN (noise)	0.0850	0.0535	0.0783	0.2714	0.0807	0.1335	99.69	179
Harris (contrast)	0.34332	0.0840	0.1668	0.2714	0.1122	0.0633	100	49
SUSAN (contrast)	0.5069	0.04168	0.4586	0.5322	0.0918	0.1982	100	141
Harris (blur)	0.0529	0.0412	0.1573	0.1831	0.0847	0.1796	99.96	49
SUSAN (blur)	2.5202	0.2822	0.8797	2.5081	0.2639	0.4809	94.92	76

Table 5.13: Comparison of the estimation errors of Harris and SUSAN corner detector

5.5 Passive Markers

Passive markers are essentially adhesive markers acting as passive sources of light. Their purpose is to increase the visibility of specific physical corners on the tanker that are not always detected by the corner detection algorithm. Not only some of the corners are undetected at times, but there is also a possibility that corners detected in the vicinity of the physical corner position could be labeled as the physical corner, leading to an error in the estimation. This problem with the labeling could be avoided by the passive markers, since passive markers ensure that the corners are always detected and hence leading to correct labeling. In other words, the use of passive markers is sought to provide an increase of the overall number of detected corners, which would in turn result in better performance of the MV system The closed loop AAR simulation executed with the Harris corner detector under nominal conditions showed that the corners numbered 3, 4, 5 and 6 in Figure 5.19 were undetected at times in the simulation. Hence an closed AAR simulation was executed with 4 passive markers at the physical corners numbered 3, 4, 5 and 6. Another AAR simulation was also executed with passive markers at all physical corner positions.

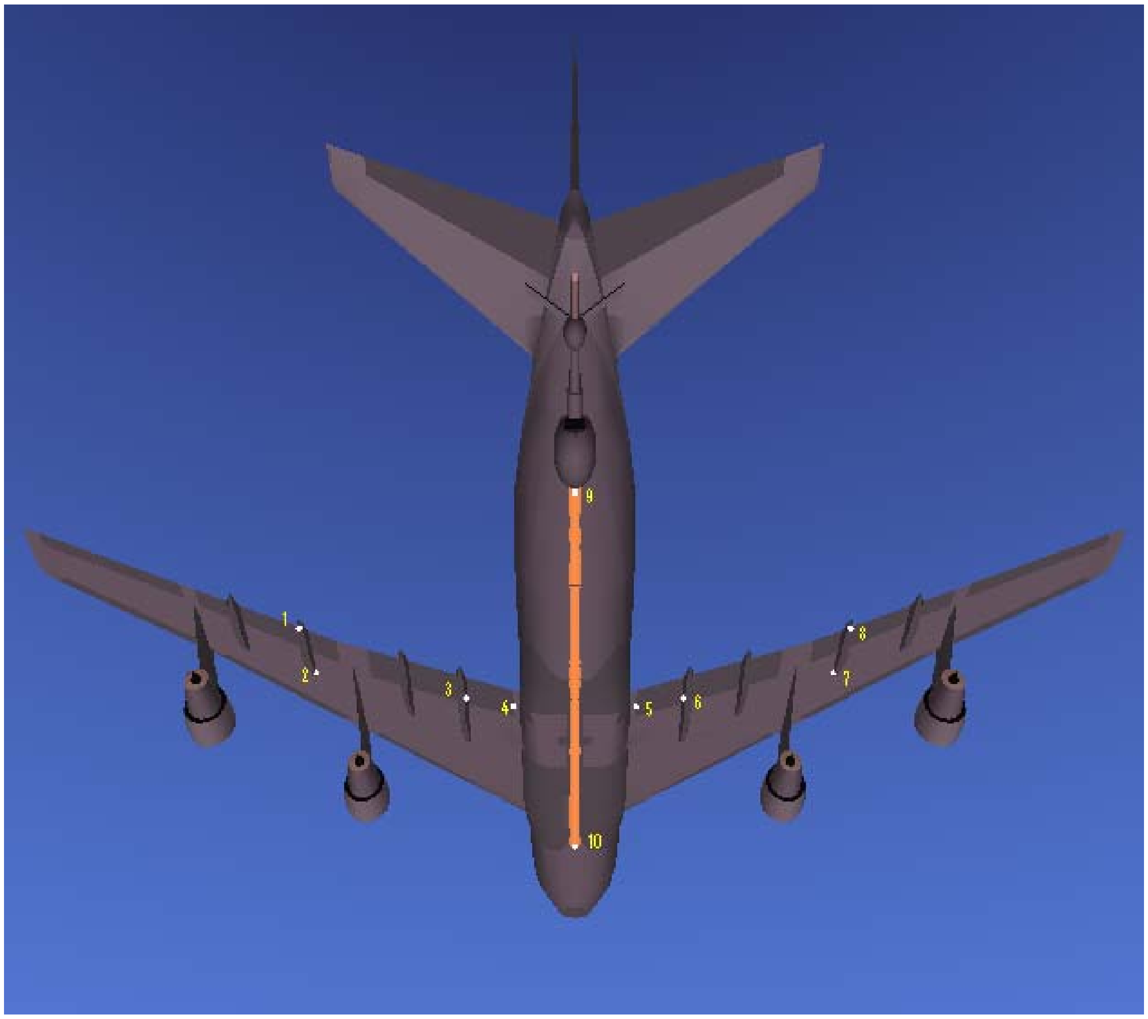


Figure 5.19: Passive markers at corner positions 3, 4, 5, and 6

Figure 5.20 shows the AAR simulation results in terms of total estimation error and number of corners detected for the three cases. The performance of the corner detector did improve with the addition of 4 passive markers. A further improvement in the performance of the corner detector was observed with passive markers at all the physical corner positions.

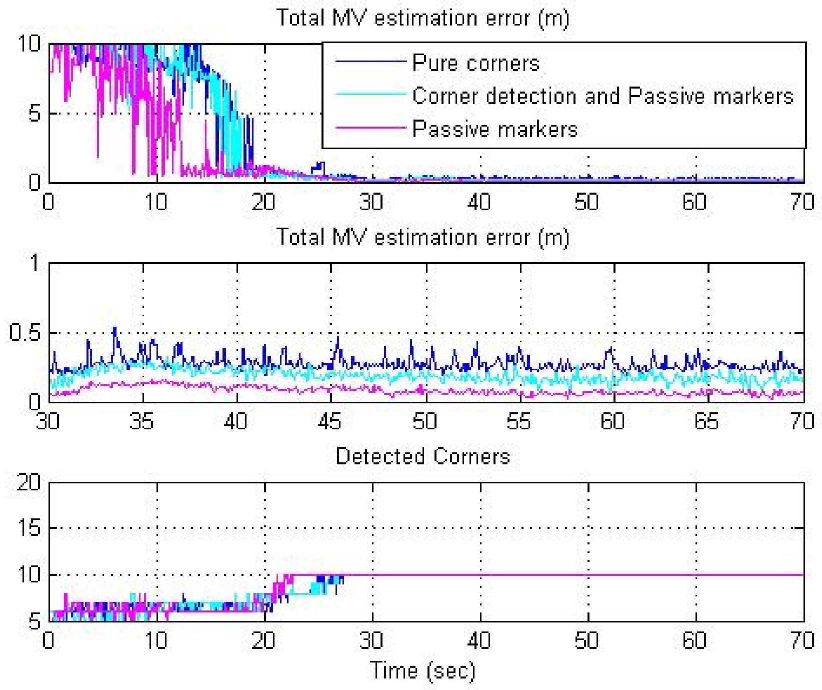


Figure 5.20: AAR simulation results with the combination of passive markers and physical corners

Table 5.14 summarizes the results for the three cases

- Case 1: Simulation with only physical corners
- Case 2: Simulation with a combination of physical corners and passive markers
- Case 3: Simulation with passive markers at all the physical corner locations

	MV rmsx	**MV rmsy**	**MV rmsz**	**BRe rmsx**	**BRe rmsy**	**BRe rmsz**	**%Avg DC**	**Max (FA)**
Case1	0.0647	0.04	0.1593	0.1751	0.0816	0.1599	100	53
Case2	0.106	0.035	0.0350	0.1751	0.0850	0.0395	100	51
Case3	0.0374	0.0266	0.0132	0.2235	0.0823	0.0497	100	46

Table 5.14: Summary of the AAR simulation results with the combination of passive markers and physical corners

The simulation results proved that the performance of the Harris corner detector did increase with the addition of the passive markers.

Chapter 6 Conclusions and Recommendations

6.1 Conclusions

The Aerial Refueling approach for an unmanned aerial vehicle is considered critical to enhance the performance of the UAV operations. Many techniques have been proposed for the acquisition of AAR capabilities to the UAV. All of these approaches proposed the use of optical markers or LEDs or beacons to determine the pose of the UAV with respect to the tanker. In this research effort, a feature extraction based approach to the AAR technique, has been considered. Corner detection algorithms have been implemented to extract the physical corners of the tanker and construct the relative position and attitude of the tanker and the UAV. All the simulation results presented in this research effort were based on the Simulink AAR scheme developed at WVU. Summarizing the experimental results of chapter 5:

- AAR simulation was successfully implemented using the feature extraction methods.
- A mixed response was obtained from the ROC curves analysis. Four images were considered for the ROC curve analysis. Two of the four images showed better performance by the Harris corner detector, while the other two images showed a better performance by the SUSAN corner detector.
- The Harris corner detector required larger computational effort when compared to the SUSAN corner detector.
- Under normal conditions the performance of the Harris and the SUSAN corner detector algorithm were similar with respect to total estimation error. The Harris corner detector had an edge over the SUSAN corner detector regarding the estimation in the 'x' direction.
- A robustness study was conducted in addition to test the corner detector stability to noise, contrast and motion blur.

- The Harris algorithm displayed robustness to all the variations induced into the images when compared to the SUSAN algorithm.
- Further, a passive markers approach has been proposed to enhance the corner detector performance.
- The performance of the corner detector did improve with the addition of passive markers.

6.2 Future work

A very interesting addition to the AAR simulation - MV system would be a pre-processing stage, where the input image to the corner detector could be analyzed to obtain information of the image regarding the contrast, brightness, noise, and certain other parameters. This information could be used to modify the corner detector parameters accordingly and hence not only improve the corner detectors performance but also of the MV system as a whole.

Template matching would also be an attractive alternative feature extraction technique to be considered for the AAR simulation.

Bibliography

[1] en.wikipedia.org, *Aerial Refueling***,** Internet.

[2] C.Bolkcom, J.D. Klaus, *Air Force Aerial Refueling methods: Flying Boom versus Probe and Drogue*, Report for Congress, 2005.

[3] M. Mammarella, *Addressing pose estimation issues for application of machine vision to UAV Autonomous Aerial Refueling,* Master's Thesis, University of Pisa, 2005.

[4] Royal Air Force, *Chapter 3: Refueling Equipment*, Internet.

[5] E. Bones, C. Bolkcom, *Unmanned Aerial Vehicles: Background and issues for Congress*, Report for Congress, 2003.

[6] M.L. Fravolini, A. Ficola, G. Campa, M.R. Napolitano, B. Seanor, *Modeling and Control Issues for Autonomous Aerial Refueling for UAVs Using a Probe-Drouge Refueling System*, Aerospace Science and Technology. Vol. 8, no. 7, pp. 611-618. Oct. 2004

[7] Mario L. Fravolini, Giampiero Campa, Antonio Ficola, Marcello R. Napolitano, Brad A. Seanor *Modeling and Control Issues for Machine Vision-Based Autonomous Aerial Refueling for UAVs*, AIAA Journal of Guidance, Control, and Dynamics, April 2005

[8] Jeffery L. Stephenson, *The Air Refueling Receiver that does not Complain*, School of Advanced Airpower studies Air University, Maxwell Air force Base, Alabama, 1998.

[9] E.M. Atkins, R.H. Miller, T.V. Pelt, K.D. Shaw, W.B. Ribbens**,** P.D. Washabaugh, *Solus: An Autonomous Aircraft for flight control and Trajectory Planning Research,* Proceedings of the American Control Conference, Philadelphia, 1998, pp. 689-693.

[10] Y. Hui, C. Zhiping, X. Shanjia, W. Shisong, *An Unmanned Air Vehicle (UAV) GPS Location and Navigation System,* Proceedings of CJMW'98, Beijing, China, August 1998, pp. 472-475.

[11] *DGPS general information*, U.S. Coast Guard Navigation Center, 2005. [Online]. Available: http://www.navcen.uscg.gov/dgps

[12] Bowers, Roshawn Elizabeth (2005). Estimation algorithm for autonomous aerial refueling using a vision based relative navigation system. Master's thesis, Texas A&M University.

[13] M Tandale, R. Bowers, J. Valasek, *Robust Trajectory Tracking Controller for Vision Based Probe and Drogue Autonomous Aerial Refueling*, in AIAA

Guidance, Navigation, and Control Conference and Exhibit, San Francisco, CA, Aug. 2005, paper AIAA-2005-5868.

[14] M. Fravolini, A Ficola, M. Napolitano, G. Campa, M. Perhinschi, *Development of Modeling and Control Tools for Aerial Refueling for UAVs*, AIAA Guidance, Navigation, and Control Conference and Exhibit, Austin, Texas, Aug. 11-14, 2003.

[15] G. Campa, M.L. Fravolini, A. Ficola, M.R. Napolitano, B. Seanor, *Autonomous Aerial Refueling for UAVs using a combined GPS-Machine Vision guidance*, AIAA Guidance, Navigation, and Control Conference and Exhibit; Providence, RI; USA; 16-19 Aug. 2004. pp. 1-11. 2004

[16] M.L. Fravolini, A. Ficola, G. Campa, M.R. Napolitano, B. Seanor, *Modeling and Control Issues for Autonomous Aerial Refueling for UAVs Using a Probe-Drouge Refueling System*, Aerospace Science and Technology. Vol. 8, no. 7, pp. 611-618. Oct. 2004

[17] L. Pollini, G. Campa, F.Giulietti, M.Innocenti, *Virtual Simulation Set-up for UAVs Aerial Refueling*, Proceedings of the 2003 AIAA Conference on Modeling and Simulation Technologies and Exhibits, Paper 2003-568211-14, Austin (TX), August 2003.

[18] L. Pollini, R. Mati, M. Innocenti, *Experimental Evaluation of Vision Algorithms for Formation Flight and Aerial Refueling*, AIAA Modeling and Simulation Technologies Conference and Exhibit; Providence, RI; August 16-19, 2004.

[19] 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: AIAA Journal of Aerospace Computing, Information and Communication*,* April 2005.

[20] S. Vendra, G. Campa, Marcello R. Napolitano, Marco Mammarella, Mario L. Fravolini, *Addressing Corner Detection Issues for Machine Vision based UAV Aerial Refueling*, Submitted to: Machine Vision and Application, October 2005.

[21] J. Valasek, J. Kimmet, D. Hughes, K. Gumman and J. Junkins., *Vision Based Sensor and Navigation System for Autonomous Aerial Refueling*, AIAA-2003-3441, 1st AIAA Unmanned Aerospace Vehicles, Systems Technologies and Operations Conference, Portsmouth, VA, 20-22 May 2002

[22] Roskam J, *Airplane Flight Dynamics and Automatic Flight Controls – Part II*, DARC Corporation, Lawrence, KS, 1994.

[23] Blake W, Gingras D.R., "Comparison of Predicted and Measured Formation Flight Interference Effect", Proceedings of the 2001 AIAA Atmospheric Flight Mechanics Conference, AIAA Paper 2001-4136, Montreal, August 2001.

[24] Gingras D.R., Player J.L., Blake W. "Static and Dynamic Wind Tunnel testing of Air Vehicles in Close Proximity", Proceedings of the AIAA Atmospheric Flight Mechanics Conference, AIAA Paper 2001-4137, Montreal, August 2001.

[25] S.M. Smith, *Feature Based Image Sequence Understanding*, D.Phil thesis, Robotics Research Group, Department of Engineering Science, Oxford University, 1992.

[26] S. Mohit, *Real Time Interactive Object Tracking,* Master's Thesis, Department of Computation, University of Manchester Institute of Science and Technology. 1998

[27] D. Parks, *Corner Detectors*, Center for Intelligent Machines. [online]

[28] F. Chabat, G.Z. Yang and D.M. Hansell, *A corner orientation detector,* Image and Vision Computing, 17-10 (1999), 761-769.

[29] D. Csetverikov, *Basic Algorithms for Digital Image Analysis: a course*, Lecture notes, Institute of Informatics, Eötvös Loránd University

[30] A. Yilmaz, *Computer Vision Systems: Lecture 4*, School of Electrical Engineering and Computer Science.

[31] S.M. Smith and J.M. Brady. *SUSAN - A New Approach to Low Level Image Processing*. Technical Report TR95SMS1c, Defense Research Agency, Farnborough, England, 1995. Also appears in IJCV'97 23(1):45-78.

[32] Noise Generation, http://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm, [online]

[33] T.S. Jebara, *3D Pose Estimation and Normalization for Face Recognition,* Bachelors Thesis McGill Center for Intelligent Machines, 1996.

[34] M. Stagg, *Robot Vision Algorithms*, [online]

[35] H. P. Moravec. *Visual Mapping by a Robot Rover*. In Proc. of the 6th International Joint Conference on Artificial Intelligence, pages 598-600, 1979.

[36] L. Kitchen and A. Rosenfeld. *Gray-Level Corner Detection*. Pattern Recognition Letters, 1:95-102, 1982.

[37] H. Wang and J. M. Brady. *Corner Detection with Subpixel Accuracy*. Technical Report OUEL-1925/92, Department of Engineering Science, University of Oxford, 1992.

[38] P.R. Beaudet. *Rotational invariant image operators*. In Proc. of the Int. Conf. on Pattern Recognition, pages 579--583, 1978

[39] R. Deriche and G. Giraudon. Accurate corner detection: An analytical study. In *Proc. 3rd Int. Conf. on Computer Vision*, pages 66--70, 1990.

[40] C.G. Harris and M. Stephens. *A combined corner and edge detector*. In 4th Alvey Vision Conference, pages 147--151, 1988.

[41] Z.Zheng, H.Wang and EKTeoh. *Analysis of Gray Level Corner Detection*. Pattern Recognition Letters, Vol. 20, pp. 149-162, 1999.

[42] J.A. Noble. *Descriptions of Image Surfaces*, D.Phil thesis, Robotics Research Group, Department of Engineering Science, Oxford University, 1989.

[43] Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. International Journal of Computer Vision 37 (2000) 151–172.

[44] Vincent, E., Lagani`ere, R.: Matching feature points in stereo pairs: A comparative study of some matching strategies. Machine Graphics & Vision **10** (2001) 237–259

[45] Molton, N., Brady, M.: Practical structure and motion from stereo when motion is unconstrained. International Journal of Computer Vision **39** (2001) 5–23

[46] Shi, J., Tomasi, C.: Good features to track. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (1994) 593–600

[47] Schmid, C., Mohr, R.: Local grayvalue invariants for image retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997) 530–535

[48] Carneiro, G., Jepson, A.D.: Local phase–based features. In: Proc. of the European Conference on Computer Vision (ECCV), Copenhagen, Denmark (2002) 282–296

[49] Knapek, M., Swain-Oropeza, R., Kriegman, D.: Selecting promising landmarks. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, USA (2000).

[50] W. Wang and R. D. Dony, *Evaluation of image corner detectors for hardware implementation*, School of Engineering, University of Guelph, Guelph, ON, Canada, Tech. Rep., 2002.

[51] D. Zhou, Y.H. Liu, X. Cai, *An Efficient and Robust Corner Detection Algorithm*, Proceedings of the 5th World Congress on Intelligent Control and Automation, June 15-19, 2004, Hangzhou, P.R. China.

[52] Air Force Online Database: *http://www.af.mil/factsheets*, (2004).

[53] Stevens, B.L., Lewis, F.L., *Aircraft Control and Simulation,* John Wiley & Sons, New York, 1987

[54] Asada H.J, Slotine J.E., *Robot Analysis and Control*, Wiley, New York, 1986

[55] Spong M.W., Vidyasagar M., *Robot Dynamics and Control*, Wiley, New York, 1989.

[56] Addington, G.A., Myatt, J.H., *Control-Surface Deflection Effects on the Innovative Control Effectors (ICE 101) Design*, Air Force Report", AFRL-VA-WP-TR-2000-3027, June 2000.

[57] B.L. Stevens, F.L. Lewis, *Aircraft Control and Simulation,* John Wiley & Sons, New York, 1987.

[58] Hutchinson. S., Hager, G., Corke, P., *A tutorial on visual servo control*, IEEE Transactions on Robotics and Automation, Vol. 12, No. 5, 1996, pp. 651-670.

[59] I. Cohen, G. Medioni, *Detecting and Tracking Moving Objects in Video from an Airborne Observer*, In DARPA Image Understanding Workshop, 1998.

[60] K. Bowyer, C. Kranenburg, and S. Dougherty, *Edge Detector Evaluation Using Empirical ROC Curves*, Computer Vision and Image Understanding, vol. 84, no. 10, pp. 77-103, 2001

[61] L. Martinez-Fonte, S. Gautama and W. Philips, *An Empirical Study on Corner Detection to Extract Buildings in Very High Resolution Satellite Images*, IEEE-ProRisc, 25-26 November 2004, Veldhoven, The Netherlands. Proceedings of ProRisc 2004, pp. 288-293.

[62] C-P Lu, G. Hager, and E. Mjolsness, "Fast and Globally Convergent Pose Estimation From Video Images", IEEE Transaction on Pattern analysis and machine intelligence, vol. 22, pp. 610 - 622, 2000.

[63] http://www.af.mil/photos,*U.S Air Force Photo*, Internet.

[64] http://www.faa.gov/education_research, *Federal Aviation Administration*, Internet.

[65] http://www.navy.mil**,** *U.S. Navy Photo*, Internet.

[66] *www.lietadle.com*, *Multi point refueling system*, Internet.

[67] www.northropgrumman.com, *Unmanned Aerial Vehicles*, Internet.

[68] uav.wff.nasa.gov, *Unmanned Aerial Vehicles,* Internet.

[69] www.uavforum.com, *Unmanned Aerial Vehicles,* Internet.

Appendix A

SUSAN Corner Detector

```
/* **** wrapper to call susan from simulink, S.Vendra, G.Campa, May
2005 ***** */

#define S_FUNCTION_NAME  susan
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

/*
 * Needed by susan
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <malloc.h>      /* may want to remove this line */

#define  exit_error(IFB,IFC) { fprintf(stderr,IFB,IFC); exit(0); }

#define  FTOI(a) ( (a) < 0 ? ((int)(a-0.5)) : ((int)(a+0.5)) )
#define SEVEN_SUPP          /* size for non-max corner suppression;
SEVEN_SUPP or FIVE_SUPP */
#define MAX_CORNERS   500  /*maximum number of corners*/

typedef  unsigned char uchar;
typedef  struct {int x,y,info, dx, dy, I;} CORNER_LIST[MAX_CORNERS];

void setup_brightness_lut(bp,thresh,form)
  uchar **bp;
  int   thresh, form;
{
int   k;
float temp;

  *bp=(unsigned char *)malloc(516);
  *bp=*bp+258;

  for(k=-256;k<257;k++)
  {
    temp=((float)k)/((float)thresh);
    temp=temp*temp;
    if (form==6)
      temp=temp*temp*temp;
    temp=100.0*exp(-temp);
    *(*bp+k)= (uchar)temp;
  }
}

/* }}} */
/* {{{ susan(in,r,sf,max_no,corner_list) */
/*******************************************************************
 in            : unsigned char **, contains pixel array (get_image)
 r              : ??? doesn't seem important as input or output...used
internally
 bp             : brightness pointer  (setup_brightness_lut)
 max_no_corners: max corners that can be found
 corner_list    : max corners array of structure
                  x,y,info,dx,dy,i
 x_size         : # horizontal pixels  (get_image)
 y_size         : # vertical pixels    (get_image)
*****************************************************************/
susan_corners(in,r,bp,max_no,corner_list,x_size,y_size)
  uchar       *in, *bp;
  int         *r, max_no, x_size, y_size;
  CORNER_LIST corner_list;
{
int   n,x,y,sq,xx,yy,
      i,j,*cgx,*cgy;
float divide;
uchar c,*p,*cp;

  /* initialize  r to zeros array (size of pic) */
  memset (r,0,x_size * y_size * sizeof(int));

  cgx=(int *)malloc(x_size*y_size*sizeof(int));
  cgy=(int *)malloc(x_size*y_size*sizeof(int));

  for (i=5;i<y_size-5;i++)
    for (j=5;j<x_size-5;j++) {
        n=100;
        p=in + (i-3)*x_size + j - 1;
        cp=bp + in[i*x_size+j];

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-3;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-5;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-6;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
      if (n<max_no){
        p+=2;
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p);
      if (n<max_no){
        p+=x_size-6;

        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p);
      if (n<max_no){
        p+=x_size-5;

        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p);
      if (n<max_no){
        p+=x_size-3;

        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p++);
      if (n<max_no){
        n+=*(cp-*p);

        if (n<max_no)
        {
            x=0;y=0;
            p=in + (i-3)*x_size + j - 1;

            c=*(cp-*p++);x-=c;y-=3*c;
            c=*(cp-*p++);y-=3*c;
            c=*(cp-*p);x+=c;y-=3*c;
            p+=x_size-3;

            c=*(cp-*p++);x-=2*c;y-=2*c;
            c=*(cp-*p++);x-=c;y-=2*c;
            c=*(cp-*p++);y-=2*c;
            c=*(cp-*p++);x+=c;y-=2*c;
            c=*(cp-*p);x+=2*c;y-=2*c;
```

```
p+=x_size-5;

c=*(cp-*p++);x-=3*c;y-=c;
c=*(cp-*p++);x-=2*c;y-=c;
c=*(cp-*p++);x-=c;y-=c;
c=*(cp-*p++);y-=c;
c=*(cp-*p++);x+=c;y-=c;
c=*(cp-*p++);x+=2*c;y-=c;
c=*(cp-*p);x+=3*c;y-=c;
p+=x_size-6;

c=*(cp-*p++);x-=3*c;
c=*(cp-*p++);x-=2*c;
c=*(cp-*p);x-=c;
p+=2;
c=*(cp-*p++);x+=c;
c=*(cp-*p++);x+=2*c;
c=*(cp-*p);x+=3*c;
p+=x_size-6;

c=*(cp-*p++);x-=3*c;y+=c;
c=*(cp-*p++);x-=2*c;y+=c;
c=*(cp-*p++);x-=c;y+=c;
c=*(cp-*p++);y+=c;
c=*(cp-*p++);x+=c;y+=c;
c=*(cp-*p++);x+=2*c;y+=c;
c=*(cp-*p);x+=3*c;y+=c;
p+=x_size-5;

c=*(cp-*p++);x-=2*c;y+=2*c;
c=*(cp-*p++);x-=c;y+=2*c;
c=*(cp-*p++);y+=2*c;
c=*(cp-*p++);x+=c;y+=2*c;
c=*(cp-*p);x+=2*c;y+=2*c;
p+=x_size-3;

c=*(cp-*p++);x-=c;y+=3*c;
c=*(cp-*p++);y+=3*c;
c=*(cp-*p);x+=c;y+=3*c;

xx=x*x;
yy=y*y;
sq=xx+yy;
if ( sq > ((n*n)/2) )
{
  if(yy<xx) {
    divide=(float)y/(float)abs(x);
    sq=abs(x)/x;
    sq=*(cp-in[(i+FTOI(divide))*x_size+j+sq]) +
       *(cp-in[(i+FTOI(2*divide))*x_size+j+2*sq]) +
       *(cp-in[(i+FTOI(3*divide))*x_size+j+3*sq]);}
  else {
    divide=(float)x/(float)abs(y);
    sq=abs(y)/y;
    sq=*(cp-in[(i+sq)*x_size+j+FTOI(divide)]) +
       *(cp-in[(i+2*sq)*x_size+j+FTOI(2*divide)]) +
       *(cp-in[(i+3*sq)*x_size+j+FTOI(3*divide)]);}
```

```
              if(sq>290){
                r[i*x_size+j] = max_no-n;
                cgx[i*x_size+j] = (51*x)/n;
                cgy[i*x_size+j] = (51*y)/n;}
            }
    }
}}}}}}}}}}}}}}}}}}

  /* to locate the local maxima */
  n=0;
  for (i=5;i<y_size-5;i++)
    for (j=5;j<x_size-5;j++) {
       x = r[i*x_size+j];
       if (x>0)  {
          /* 5x5 mask */
#ifdef FIVE_SUPP
          if (
              (x>r[(i-1)*x_size+j+2]) &&
              (x>r[(i  )*x_size+j+1]) &&
              (x>r[(i  )*x_size+j+2]) &&
              (x>r[(i+1)*x_size+j-1]) &&
              (x>r[(i+1)*x_size+j  ]) &&
              (x>r[(i+1)*x_size+j+1]) &&
              (x>r[(i+1)*x_size+j+2]) &&
              (x>r[(i+2)*x_size+j-2]) &&
              (x>r[(i+2)*x_size+j-1]) &&
              (x>r[(i+2)*x_size+j  ]) &&
              (x>r[(i+2)*x_size+j+1]) &&
              (x>r[(i+2)*x_size+j+2]) &&
              (x>=r[(i-2)*x_size+j-2]) &&
              (x>=r[(i-2)*x_size+j-1]) &&
              (x>=r[(i-2)*x_size+j  ]) &&
              (x>=r[(i-2)*x_size+j+1]) &&
              (x>=r[(i-2)*x_size+j+2]) &&
              (x>=r[(i-1)*x_size+j-2]) &&
              (x>=r[(i-1)*x_size+j-1]) &&
          (x>=r[(i-1)*x_size+j  ]) &&
          (x>=r[(i-1)*x_size+j+1]) &&
          (x>=r[(i  )*x_size+j-2]) &&
          (x>=r[(i  )*x_size+j-1]) &&
          (x>=r[(i+1)*x_size+j-2]) )
#endif
#ifdef SEVEN_SUPP
          if (
                (x>r[(i-3)*x_size+j-3]) &&
                (x>r[(i-3)*x_size+j-2]) &&
                (x>r[(i-3)*x_size+j-1]) &&
                (x>r[(i-3)*x_size+j  ]) &&
                (x>r[(i-3)*x_size+j+1]) &&
                (x>r[(i-3)*x_size+j+2]) &&
                (x>r[(i-3)*x_size+j+3]) &&

                (x>r[(i-2)*x_size+j-3]) &&
                (x>r[(i-2)*x_size+j-2]) &&
                (x>r[(i-2)*x_size+j-1]) &&
                (x>r[(i-2)*x_size+j  ]) &&
                (x>r[(i-2)*x_size+j+1]) &&
                (x>r[(i-2)*x_size+j+2]) &&
                (x>r[(i-2)*x_size+j+3]) &&

                (x>r[(i-1)*x_size+j-3]) &&
                (x>r[(i-1)*x_size+j-2]) &&
                (x>r[(i-1)*x_size+j-1]) &&
                (x>r[(i-1)*x_size+j  ]) &&
                (x>r[(i-1)*x_size+j+1]) &&
                (x>r[(i-1)*x_size+j+2]) &&
                (x>r[(i-1)*x_size+j+3]) &&

                (x>r[(i)*x_size+j-3]) &&
                (x>r[(i)*x_size+j-2]) &&
                (x>r[(i)*x_size+j-1]) &&
                (x>=r[(i)*x_size+j+1]) &&
                (x>=r[(i)*x_size+j+2]) &&
                (x>=r[(i)*x_size+j+3]) &&

                (x>=r[(i+1)*x_size+j-3]) &&
                (x>=r[(i+1)*x_size+j-2]) &&
                (x>=r[(i+1)*x_size+j-1]) &&
                (x>=r[(i+1)*x_size+j  ]) &&
                (x>=r[(i+1)*x_size+j+1]) &&
                (x>=r[(i+1)*x_size+j+2]) &&
                (x>=r[(i+1)*x_size+j+3]) &&

                (x>=r[(i+2)*x_size+j-3]) &&
                (x>=r[(i+2)*x_size+j-2]) &&
                (x>=r[(i+2)*x_size+j-1]) &&
                (x>=r[(i+2)*x_size+j  ]) &&
                (x>=r[(i+2)*x_size+j+1]) &&
                (x>=r[(i+2)*x_size+j+2]) &&
                (x>=r[(i+2)*x_size+j+3]) &&

                (x>=r[(i+3)*x_size+j-3]) &&
                (x>=r[(i+3)*x_size+j-2]) &&
                (x>=r[(i+3)*x_size+j-1]) &&
                (x>=r[(i+3)*x_size+j  ]) &&
                (x>=r[(i+3)*x_size+j+1]) &&
                (x>=r[(i+3)*x_size+j+2]) &&
                (x>=r[(i+3)*x_size+j+3]) )
#endif
{
corner_list[n].info=0;
corner_list[n].x=j;
corner_list[n].y=i;
corner_list[n].dx=cgx[i*x_size+j];
corner_list[n].dy=cgy[i*x_size+j];
corner_list[n].I=in[i*x_size+j];
n++;
if(n==MAX_CORNERS){
      fprintf(stderr,"Too many corners.\n");
      exit(1);
         }}}}
corner_list[n].info=7;
```

```
free(cgx);
free(cgy);

}


/*====================*
 * S-function methods *
 *====================*/


#define MDL_CHECK_PARAMETERS
static void mdlCheckParameters(SimStruct *S)
{
    /* Basic check : All parameters must be real positive vectors
*/
    real_T *pr;

    int_T  i, el, nEls;
    for (i = 0; i < 4; i++) {
        if (mxIsEmpty(    ssGetSFcnParam(S,i)) || mxIsSparse(
ssGetSFcnParam(S,i)) ||
            mxIsComplex(  ssGetSFcnParam(S,i)) || !mxIsNumeric(
ssGetSFcnParam(S,i))  )
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        pr   = mxGetPr(ssGetSFcnParam(S,i));
        nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
        for (el = 0; el < nEls; el++) {
            if (!mxIsFinite(pr[el]))
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        }
    }

    /* Check number of elements in parameter: nmax
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,0)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,0));
    if ( pr[0] < 1 | pr[0] > 500 )
    { ssSetErrorStatus(S,"The size of the corner vector must be between
one and 500"); return; }

    /* Check number of elements in parameter: image size
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,1)) != 2 )      //
screen limit
    { ssSetErrorStatus(S,"The Image Size must be a 3 elements vector:
[320 400 3]"); return; }
    pr=mxGetPr(ssGetSFcnParam(S,1));
    if ( pr[0] < 1 || pr[1] < 1)
    { ssSetErrorStatus(S,"The first two Image Size elements should be
greater than zero and the third element should be always 3"); return; }
```

```
     /* Check number of elements in parameter: bt brightness threshold
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,2)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,2));
    if ( pr[0] < 1 )
    { ssSetErrorStatus(S,"The brightness threshold must be greater than
zero"); return; }


     /* Check number of elements in parameter: sampling time
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,3)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,3));
    if ( pr[0] < 0 )
    { ssSetErrorStatus(S,"Sampling Time cannot be negative"); return; }

}


/* Function: mdlInitializeSizes
===============================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-
function
 *    block's characteristics (number of inputs, outputs, states,
etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    real_T *nmax, *imgsize;
    //int max=40;

    nmax=mxGetPr(ssGetSFcnParam(S,0));                // Maximum number
of markers
    imgsize=mxGetPr(ssGetSFcnParam(S,1));             // Image size
vector

    ssSetNumSFcnParams(S,4);                          /* number of
expected parameters        */

    /* Check the number of parameters and then calls mdlCheckParameters
to see if they are ok */
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
    { mdlCheckParameters(S); if (ssGetErrorStatus(S) != NULL) return; }
else return;

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, (imgsize[0]*imgsize[1]));
    ssSetInputPortDataType(S,0,SS_UINT8);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, true); /*direct input signal
access*/

    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, nmax[0]*2);/*max is the maximum number
of corners */

    ssSetNumSampleTimes(S, 0);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S,imgsize[0]*imgsize[1]);                         /*
number int_T work vector elements    */
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}



/* Function: mdlInitializeSampleTimes
=========================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    real_T *t;
    t=mxGetPr(ssGetSFcnParam(S,3));

    ssSetSampleTime(S, 0, *t);
    ssSetOffsetTime(S, 0, 0);

}

#define MDL_START  /* Change to #undef to remove function */
  /* Function: mdlStart
=======================================================
   * Abstract:
   *    This function is called once at start of model execution. If
you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
  }
```

```
/* Function: mdlOutputs
=======================================================
 * Abstract:
 *    In this function, you compute the outputs of your S-function
 *    block. Generally outputs are placed in the output vector,
ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    uint8_T *u = (uint8_T*) ssGetInputPortSignal(S,0);
    real_T       *y  = ssGetOutputPortSignal(S,0);
    real_T    *nmax  = mxGetPr(ssGetSFcnParam(S,0));
    real_T *imgsize  = mxGetPr(ssGetSFcnParam(S,1));
    real_T *bt       = mxGetPr(ssGetSFcnParam(S,2));

    unsigned char *bp;
    int *r = ssGetIWork(S);
    int nn1=0,nn2=0,nn3=0,nn4=1,area_max=1850,N=(int)nmax[0];
    CORNER_LIST corner_list;

    setup_brightness_lut(&bp,(int)bt[0],6);

   // printf("bt= %d, bp[3]= %d, m= %d, n= %d
\n",(int)bt[0],bp[3],(int) imgsize[0],(int) imgsize[1]);

    susan_corners(u,r,bp,area_max,corner_list,(int) imgsize[0],(int)
imgsize[1]);

    while(corner_list[nn1].info!=7){
        y[nn2]=(real_T) corner_list[nn1].y;
        nn2=nn2+2;
        nn1++;
    }

    while(corner_list[nn3].info!=7){
        y[nn4]=(real_T) corner_list[nn3].x;
        nn4=nn4+2;
        nn3++;
    }
    while(nn1<N){
        y[nn2]=-1;
        nn2=nn2+2;
        nn1++;
    }

    while(nn3<N){
        y[nn4]=-1;
        nn4=nn4+2;
        nn3++;
    }
}



#define MDL_UPDATE  /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
```

```
  /* Function: mdlUpdate
======================================================
   * Abstract:
   *    This function is called once for every major integration time
step.
   *    Discrete states are typically updated here, but this function
is useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */



#define MDL_DERIVATIVES  /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives
=================================================
   * Abstract:
   *    In this function, you compute the S-function block's
derivatives.
   *    The derivatives are placed in the derivative vector,
ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */



/* Function: mdlTerminate
=====================================================
 * Abstract:
 *    In this function, you should perform any actions that are
necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}


/*======================================================*
 * See sfuntmpl_doc.c for the optional S-function methods *
 *======================================================*/

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
```

```
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Appendix B

Harris Corner Detector

```
#define S_FUNCTION_NAME  harris
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

/*
 * Needed by sfunHarris.c
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <malloc.h>
# define MINUS_INFINITY -32767

// Function for performing the 1D convolution
 conv1d(double *u,double *v,double *w1,double *w,int d1,int d2,int dim)
{
    int kk,ii,pos,i;
    memset(w1,0,dim*sizeof(double));
    pos=d1;
    d1=d2;
    d2=pos;

    for (kk=0;kk<dim;kk++){
     for (ii=0;ii<kk+1;ii++)
     { if (ii<d1)
       {
       pos=kk-ii;
       if(pos<d2)
       {
       *(w1+kk)=*(w1+kk)+*(u+ii)*(*(v+pos));
       }
       }
       else break;
     }
    }
    for(i=0;i<(int)d2;i++){
        *(w+i) = *(w1+i+(((int)d1-1)/2));
    }

}


// Function for performing the Gaussian smoothning
gauss(double *imd,double sigma,int imx, int imy,int padding,double
*outd)
{
    double *tempd;
    int tempx,tempy;
    int ii,jj;
    double *in_scand,*out_scand;
    double q,qq,qqq,b0,b1,b2,b3,B;
    double *tempd1,*tempd2;
    double bc;
    int riga;

    q = 1.31564 * (sqrt(1 + 0.490811 * sigma*sigma) - 1);
    qq = q*q;
    qqq = qq*q;
    b0 = 1.0/(1.57825 + 2.44413*q + 1.4281*qq + 0.422205*qqq);
    b1 = (2.44413*q + 2.85619*qq + 1.26661*qqq)*b0;
    b2 = (-1.4281*qq - 1.26661*qqq)*b0;
    b3 = 0.422205*qqq*b0;
    B = 1.0 - (b1 + b2 + b3);

    tempx=imx+2*padding;
    tempy=imy+2*padding;

    tempd  = malloc(tempx*tempy*sizeof(double));
    tempd1 = malloc(tempx*tempy*sizeof(double));
    tempd2 = malloc(tempx*tempy*sizeof(double));

    for (jj=0;jj<imy;jj++)
        {
            out_scand=tempd+tempx*(jj+padding)+padding;
            in_scand=imd+imx*jj;
                for (ii=0;ii<imx;ii++)
                    {
                        *out_scand=*in_scand;
                        out_scand++;
                        in_scand++;
                    }
        }

    for (jj=0;jj<tempy;jj++)
        {
            bc=*(tempd+tempx*jj);

*(tempd1+tempx*jj+0)=B**(tempd+jj*tempx+0)+b1*bc+b2*bc+b3*bc;

*(tempd1+tempx*jj+1)=B**(tempd+jj*tempx+1)+b1*bc+b2*bc+b3*bc;

*(tempd1+tempx*jj+2)=B**(tempd+jj*tempx+2)+b1*bc+b2*bc+b3*bc;
            for (ii=3;ii<tempx;ii++)
                {

*(tempd1+tempx*jj+ii)=B**(tempd+tempx*jj+ii)+b1**(tempd1+tempx*jj+ii-
1)+b2**(tempd1+tempx*jj+ii-2)+b3**(tempd1+tempx*jj+ii-3);
                }

            bc=*(tempd1+tempx*jj+tempx-1);
            *(tempd2+tempx*jj+tempx-1-0)=B**(tempd1+tempx*jj+tempx-1-
0)+b1*bc+b2*bc+b3*bc;
            *(tempd2+tempx*jj+tempx-1-1)=B**(tempd1+tempx*jj+tempx-1-
1)+b1*bc+b2*bc+b3*bc;
            *(tempd2+tempx*jj+tempx-1-2)=B**(tempd1+tempx*jj+tempx-1-
2)+b1*bc+b2*bc+b3*bc;
```

```
            for (ii=tempx-4;ii>=0;ii--)
                {
*(tempd2+tempx*jj+ii)=B**(tempd1+tempx*jj+ii)+b1**(tempd2+tempx*jj+ii+1
)+b2**(tempd2+tempx*jj+ii+2)+b3**(tempd2+tempx*jj+ii+3);
                }

        }


    for (ii=0;ii<tempx;ii++)
        {
            bc=*(tempd2+ii);

*(tempd1+tempx*0+ii)=B**(tempd2+tempx*0+ii)+b1*bc+b2*bc+b3*bc;

*(tempd1+tempx*1+ii)=B**(tempd2+tempx*1+ii)+b1*bc+b2*bc+b3*bc;

*(tempd1+tempx*2+ii)=B**(tempd2+tempx*2+ii)+b1*bc+b2*bc+b3*bc;

            for (jj=3;jj<tempy;jj++)
                {
*(tempd1+tempx*jj+ii)=B**(tempd2+tempx*jj+ii)+b1**(tempd1+tempx*(jj-
1)+ii)+b2**(tempd1+tempx*(jj-2)+ii)+b3**(tempd1+tempx*(jj-3)+ii);
                }


            bc=*(tempd1+tempx*(tempy-1)+ii);
            *(tempd2+tempx*(tempy-1-0)+ii)=B**(tempd1+tempx*(tempy-1-
0)+ii)+b1*bc+b2*bc+b3*bc;
            *(tempd2+tempx*(tempy-1-1)+ii)=B**(tempd1+tempx*(tempy-1-
1)+ii)+b1*bc+b2*bc+b3*bc;
            *(tempd2+tempx*(tempy-1-2)+ii)=B**(tempd1+tempx*(tempy-1-
2)+ii)+b1*bc+b2*bc+b3*bc;


            for (jj=tempy-4;jj>=0;jj--)
                {
*(tempd2+tempx*jj+ii)=B**(tempd1+tempx*jj+ii)+b1**(tempd2+tempx*(jj+1)+
ii)+b2**(tempd2+tempx*(jj+2)+ii)+b3**(tempd2+tempx*(jj+3)+ii);
                }
        }

    for (jj=0;jj<imy;jj++)
        {
            in_scand=tempd2+tempx*(jj+padding)+padding;
            out_scand=outd+imx*jj;
            for (ii=0;ii<imx;ii++)
                {
                    *out_scand=*in_scand;
                    out_scand++;
                    in_scand++;
                }
        }
    free(tempd);
    free(tempd1);
    free(tempd2);
    return;
}


//Function for performing the dilation

dilate_img(int m,int n,int mmask,int nmask, double *input,double
*mask,double *buffer,double *output)
{
    int i,j,r,c;
    double sum,the_max;
     memset(buffer,0,(m+mmask-1)*(n+nmask-1)*sizeof(double));

     /*This for loop to pad zeros around the image*/
     /*0 0 0 0 0
       0 1 1 1 0
       0 1 1 1 0
       0 1 1 1 0
       0 0 0 0 0*/

     for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            *(buffer +((j+nmask-2)*(m+mmask-1))+(i+mmask-2)) = *(input
+((j)*m)+(i));
        }
    }

    /*This does the dilation and places the output in the 'output'
matrix*/
   for(r=0;r<m;r++){
        for(c=0;c<n;c++){
          the_max = MINUS_INFINITY;
          for(i=0;i<mmask;i++){
              for(j=0;j<nmask;j++){
                  sum = *(buffer +((c+j)*(m+mmask-1))+(r+i)) + *(mask
+(j*mmask)+i);
                  if (sum > the_max) the_max = sum;
              }
          }
          *(output+(c*m)+r) = (the_max-1);
        }
    }
}




/*====================*
 * S-function methods *
 *====================*/
#define MDL_CHECK_PARAMETERS
static void mdlCheckParameters(SimStruct *S)
{
    /* Basic check : All parameters must be real positive vectors
*/
    real_T *pr;

    int_T  i, el, nEls;
    for (i = 0; i < 3; i++) {
        if (mxIsEmpty(    ssGetSFcnParam(S,i)) || mxIsSparse(
ssGetSFcnParam(S,i)) ||
            mxIsComplex(  ssGetSFcnParam(S,i)) || !mxIsNumeric(
ssGetSFcnParam(S,i))  )
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        pr   = mxGetPr(ssGetSFcnParam(S,i));
        nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
        for (el = 0; el < nEls; el++) {
            if (!mxIsFinite(pr[el]))
                  { ssSetErrorStatus(S,"Parameters must be real finite
vectors"); return; }
        }
    }

    /* Check number of elements in parameter: nmax
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,0)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,0));
    if ( pr[0] < 1 )
    { ssSetErrorStatus(S,"The size of the corner vector must be greater
than zero"); return; }

     /* Check number of elements in parameter: image size
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,1)) != 2 )
// screen limit
    { ssSetErrorStatus(S,"The Image Size must be a 2 elements vector:
[320 400]"); return; }
    pr=mxGetPr(ssGetSFcnParam(S,1));
    if ( pr[0] < 1 || pr[1] < 1 )
    { ssSetErrorStatus(S,"The first two Image Size elements should be
greater than zero and the third element should be always 3"); return; }

     /* Check number of elements in parameter: Threshold value
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,2)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,2));
    if ( pr[0] < 0 )
    { ssSetErrorStatus(S,"The Thresh value must be a scalar"); return;
}
```

```
     /* Check number of elements in parameter: sampling time
*/
    if ( mxGetNumberOfElements(ssGetSFcnParam(S,3)) != 1 )
    { ssSetErrorStatus(S,"The parameter must be a scalar"); return; }

    /* get the basic parameters and check them
*/
    pr=mxGetPr(ssGetSFcnParam(S,3));
    if ( pr[0] < 0 )
    { ssSetErrorStatus(S,"Sampling Time cannot be negative"); return; }

}


/* Function: mdlInitializeSizes
===============================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-
function
 *    block's characteristics (number of inputs, outputs, states,
etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    double *nmax, *imgsize;

    nmax    = mxGetPr(ssGetSFcnParam(S,0));           // Maximum number
of markers
    imgsize = mxGetPr(ssGetSFcnParam(S,1));           // Image size
vector


    ssSetNumSFcnParams(S,4);                          /* number of
expected parameters        */

    /* Check the number of parameters and then calls mdlCheckParameters
to see if they are ok */
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
    { mdlCheckParameters(S); if (ssGetErrorStatus(S) != NULL) return; }
else return;

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, ((int)imgsize[0]*(int)imgsize[1]));
    ssSetInputPortDataType(S,0,SS_UINT8);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, true);     /*direct input
signal access*/


    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, nmax[0]*2);
    ssSetOutputPortDataType(S,0,SS_DOUBLE);

    ssSetNumSampleTimes(S, 0);
```

```
ssSetNumRWork(S,((int)imgsize[0]+2)+((int)imgsize[1]+2)+(int)imgsize[0]
+(int)imgsize[0]+(int)imgsize[0]*(int)imgsize[1]+(int)imgsize[1]+(int)i
mgsize[1]+(int)imgsize[0]*(int)imgsize[1]+(int)imgsize[0]*(int)imgsize[
1]+(int)imgsize[0]*(int)imgsize[1]+(int)imgsize[0]*(int)imgsize[1]+(int
)imgsize[0]*(int)imgsize[1]+(int)imgsize[0]*(int)imgsize[1]+(int)imgsiz
e[0]*(int)imgsize[1]+(int)imgsize[0]*(int)imgsize[1]+((int)imgsize[0]+3
-1)*((int)imgsize[1]+3-1) );
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, 0);
}



/* Function: mdlInitializeSampleTimes
=========================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    real_T *t;
    t=mxGetPr(ssGetSFcnParam(S,3));

    ssSetSampleTime(S, 0, t[0]);
    ssSetOffsetTime(S, 0, 0 );

}

#define MDL_START  /* Change to #undef to remove function */
  /* Function: mdlStart
=======================================================
   * Abstract:
   *    This function is called once at start of model execution. If
you
   *    have states that should be initialized once, this is the place
   *    to do it.
   */
  static void mdlStart(SimStruct *S)
  {
  }


/* Function: mdlOutputs
=======================================================
 * Abstract:
 *    In this function, you compute the outputs of your S-function
 *    block. Generally outputs are placed in the output vector,
ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid)
```

```
{
    const uint8_T *u1 = (const uint8_T*) ssGetInputPortSignal(S,0);
    real_T       *y   = ssGetOutputPortSignal(S,0);     // Output
    real_T    *nmax   = mxGetPr(ssGetSFcnParam(S,0));   // Max number
of corners
    real_T *imgsize   = mxGetPr(ssGetSFcnParam(S,1));   // Image Size
(2D)
    real_T *thresh    = mxGetPr(ssGetSFcnParam(S,2));   // Threshold
value

    double *mptr,*rows,*cols;
    int i,j,k=0,mb=3,nb=3,mc,nc;
    double eps = 2.2204e-016;
    double sigma = 5;
    int padding = 15;

    int d1,d2=3,kk,j1;
    int k1=0,k2=0,k3=0,k4=0,i1,i2;
    int dim1 = (int)imgsize[0]+2,dim2 = (int)imgsize[1]+2;

    // usx and vx are the derivative masks in the x direction
    double usx[3][1] = {-1.4142,
                        -1.4142,
                        -1.4142};
    double vx[1][3] = { 0.7071,0,-0.7071};
    double *usxptr = &usx;
    double *vxptr = &vx;
    // usy and vy are the derivative masks in the y direction
    double usy[3][1] = {-1.7321,
                         0,
                         1.7321};
    double vy[1][3] = { 0.5774,0.5774,0.5774};
    double *usyptr = &usy;
    double *vyptr = &vy;
    double mask[3][3] = {1,1,1,
                         1,1,1,
                         1,1,1};
    int m1,n1;

    double *buffer1   = ssGetRWork(S);
    double *buffer2   = &buffer1[(int)imgsize[0]+2];
    double *temp1     = &buffer2[(int)imgsize[1]+2];
    double *temp2     = &temp1[(int)imgsize[0]];
    double *y3        = &temp2[(int)imgsize[0]];
    double *temp3     = &temp2[(int)imgsize[0]*(int)imgsize[1]];
    double *temp4     = &temp3[(int)imgsize[1]];
    double *ixptr     = &temp4[(int)imgsize[1]]; // pointer pointing to
the image derivative in the x-direction
    double *iyptr     = &ixptr[(int)imgsize[0]*(int)imgsize[1]];//
pointer pointing to the image derivative in the y-direction
    double *ixyptr    = &iyptr[(int)imgsize[0]*(int)imgsize[1]]; //
pointer pointing to the product of the above two image derivatives
    double *ixxptr    = &ixyptr[(int)imgsize[0]*(int)imgsize[1]]; //
pointer to the smoothed image in the x-direction
    double *iyyptr    = &ixxptr[(int)imgsize[0]*(int)imgsize[1]]; //
pointer to the smoothed image in the y-direction
    double *ixy2ptr   = &iyyptr[(int)imgsize[0]*(int)imgsize[1]];
    double *cim1ptr   = &ixy2ptr[(int)imgsize[0]*(int)imgsize[1]]; //
pointer to the harris corner function
    double *outputptr = &cim1ptr[(int)imgsize[0]*(int)imgsize[1]]; //
pointer to the dilated image
    double *buffer3   =  &outputptr[(int)imgsize[0]*(int)imgsize[1]];

   // Image derivatives in the x-direction
    for(j=0;j<(int)imgsize[1];j++){

for(i1=k1,i2=0;i1<k1+(int)imgsize[0],i2<(int)imgsize[0];i1++,i2++){
           *(temp1 + i2) = (double)*(u1 + i1);
            *(temp2 + i2) = 0;
        }
        conv1d(usxptr,temp1,buffer1,temp2,(int)imgsize[0],d2,dim1);

for(i1=k1,i2=0;i1<k1+(int)imgsize[0],i2<(int)imgsize[0];i1++,i2++){
            *(y3 + i1) = *(temp2 + i2);
        }
         k1=k1+(int)imgsize[0];
    }


    for(j1=0;j1<(int)imgsize[0];j1++){
        for(i2=0;i2<(int)imgsize[1];i2++){
             *(temp3 + i2) = *(y3 + i2*(int)imgsize[0]+k2);
            *(temp4 + i2) = 0;
        }
        conv1d(vxptr,temp3,buffer2,temp4,(int)imgsize[1],d2,dim2);

       for(i2=0;i2<(int)imgsize[1];i2++){
            *(ixptr + i2*(int)imgsize[0]+k2) = *(temp4 + i2);
        }
        k2=k2+1;
    }
    // ixptr is the derivative of the image in the x-direction


    memset(y3,0,(int)imgsize[0]*(int)imgsize[1]*sizeof(double));
    memset(temp1,0,(int)imgsize[0]*sizeof(double));
    memset(temp3,0,(int)imgsize[1]*sizeof(double));

    // Image derivatives in the y-direction
     for(j=0;j<(int)imgsize[1];j++){

for(i1=k3,i2=0;i1<k3+(int)imgsize[0],i2<(int)imgsize[0];i1++,i2++){
             *(temp1 + i2) = (double)*(u1 + i1);
            *(temp2 + i2) = 0;
        }
        conv1d(usyptr,temp1,buffer1,temp2,(int)imgsize[0],d2,dim1);

for(i1=k3,i2=0;i1<k3+(int)imgsize[0],i2<(int)imgsize[0];i1++,i2++){
           *(y3 + i1) = *(temp2 + i2);
        }
         k3=k3+(int)imgsize[0];
    }
    for(j1=0;j1<(int)imgsize[0];j1++){
        for(i2=0;i2<(int)imgsize[1];i2++){
            *(temp3 + i2) = *(y3 + i2*(int)imgsize[0]+k4);
            *(temp4 + i2) = 0;
        }
        conv1d(vyptr,temp3,buffer2,temp4,(int)imgsize[1],d2,dim2);
        for(i2=0;i2<(int)imgsize[1];i2++){
            *(iyptr + i2*(int)imgsize[0]+k4) = *(temp4 + i2);
        }
        k4=k4+1;
    }

    // iyptr is the image derivative in the y-direction

    for(i=0;i<(int)imgsize[0];i++){
        for(j=0;j<(int)imgsize[1];j++){
            *(ixyptr + (j*(int)imgsize[0])+i) = (*(ixptr +
(j*(int)imgsize[0])+i))*(*(iyptr + (j*(int)imgsize[0])+i));
          }
    }
    // ixyptr is the product of the ixptr and iyptr
    for(i=0;i<(int)imgsize[0];i++){
        for(j=0;j<(int)imgsize[1];j++){
            *(ixptr + (j*(int)imgsize[0])+i) = (*(ixptr +
(j*(int)imgsize[0])+i))*(*(ixptr + (j*(int)imgsize[0])+i));
            *(iyptr + (j*(int)imgsize[0])+i) = (*(iyptr +
(j*(int)imgsize[0])+i))*(*(iyptr + (j*(int)imgsize[0])+i));

        }
    }
    // ixptr and iyptr are the respective squares of the ixptr and the
iyptr now
    // ixptr,iyprt and ixyptr are the squared image derivatives

    //Smoothning the squared image derivatives
    gauss(ixptr,sigma,(int)imgsize[0],(int)imgsize[1],padding,ixxptr);
    gauss(iyptr,sigma,(int)imgsize[0],(int)imgsize[1],padding,iyyptr);

gauss(ixyptr,sigma,(int)imgsize[0],(int)imgsize[1],padding,ixy2ptr);

    // cim1ptr is the harris corner measure
     for(i=0;i<(int)imgsize[0];i++){
        for(j=0;j<(int)imgsize[1];j++){
            *(cim1ptr + (j*(int)imgsize[0])+i) = (((*(ixxptr +
(j*(int)imgsize[0])+i))*(*(iyyptr + (j*(int)imgsize[0])+i)))-
((*(ixy2ptr + (j*(int)imgsize[0])+i))*(*(ixy2ptr +
(j*(int)imgsize[0])+i))))/(*(ixxptr + (j*(int)imgsize[0])+i)+*(iyyptr +
(j*(int)imgsize[0])+i)+eps);
        }
    }

    // Extract local maxima by performing a grey scale morphological
dilation

dilate_img((int)imgsize[0],(int)imgsize[1],mb,nb,cim1ptr,mask,buffer3,o
utputptr);
```

```
     // Initializing the output vector
    for(i=0;i<(int)nmax[0]*2;i++){
        *(y+i) = (double)(-1);
    }

    // Finding points in the corner strength image that
    // match the dilated image and are also greater than the threshold.

     for(i=0;i<(int)imgsize[0];i++){
        for(j=0;j<(int)imgsize[1];j++){

if((*(cim1ptr+(j*(int)imgsize[0])+i)>(float)thresh[0])&&(*(cim1ptr+(j*(
int)imgsize[0])+i)==*(outputptr+(j*(int)imgsize[0])+i)))
            {

               *(y+k)   = (double)(j+1);
               *(y+k+1) = (double)(i+1);
               k=k+2;
            }

        }
    }

}



#define MDL_UPDATE  /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate
=======================================================
   * Abstract:
   *    This function is called once for every major integration time
step.
   *    Discrete states are typically updated here, but this function
is useful
   *    for performing any tasks that should only take place once per
   *    integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */



#define MDL_DERIVATIVES  /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives
==================================================
   * Abstract:
   *    In this function, you compute the S-function block's
derivatives.
   *    The derivatives are placed in the derivative vector,
ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */



/* Function: mdlTerminate
=====================================================
 * Abstract:
 *    In this function, you should perform any actions that are
necessary
 *    at the termination of a simulation.  For example, if memory was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{

}


/*======================================================*
 * See sfuntmpl_doc.c for the optional S-function methods *
 *======================================================*/

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```