A Vision-Based Algorithm for UAV State Estimation During Vehicle Recovery

By

William Robert Burns

Submitted to the graduate degree program in Aerospace Engineering and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Date Defended: August 18, 2011

The Thesis Committee for William Robert Burns

certifies that this is the approved version of the following thesis:

A Vision-Based Algorithm for UAV State Estimation During Vehicle Recovery

Chairperson Shahriar Keshmiri

Date approved: August 18, 2011

Abstract

A computer vision-based algorithm for Unmanned Aerial Vehicle state estimation during vehicle recovery is presented. The algorithm is intended to be used to augment or back up Global Positioning System as the primary means of navigation during vehicle recovery for UAVs. The method requires a clearly visible recovery target with markers placed on the corners in addition to known target geometry. The algorithm uses clustering techniques to identify the markers, a Canny Edge detector and a Hough Transform to verify these markers actually lie on the recovery target, an optimizer to match the detected markers with coordinates in three-space, a non-linear transformation and projection solver to observe the position and orientation of the camera, and an Extended Kalman Filter (EKF) to improve the tracking of the state estimate. While it must be acknowledged that the resolution of the test images used is much higher than the resolution of images used in previous algorithms and that the images used to test this algorithm are either synthetic or taken in static conditions, the algorithm presented does give much better state estimates than previously-developed vision systems.

Acknowledgments

I would like to first like to acknowledge Dr. Shah Keshmiri for his unwavering support, patience, and generosity over the last few years. I would like to thank Dr. Dave Downing for his patience and insight not only with this Thesis, but with all of our interactions since my freshman year. I would like to thank Dr. Brian Potetz for exceptionally high-quality education in EECS 741, the class where this project got started. Thanks also go to Dr. Rick Hale and Mark Ewing for being willing to provide indirect funding for this activity despite the ever-pressing needs of the Meridian.

I would like to thank my friends for being so patient with me during these last few months of my being, well, pretty lame. I would like to thank my family for all of the many things they have done for me since I started college. I want to thank my amazing girlfriend, Eliza, for being constantly supportive and encouraging, and always a source of positivity. Finally, I want to thank the good Lord for all of His many blessings, especially those that have helped me get to this point.

Table of Contents

Page #

List of Symbols

Symbol	Description	Units
AR	Image aspect ratio, w/h	-
b_x	Camera body x-coordinate in world coordinate system	ft. or in.
b_y	Camera body y-coordinate in world coordinate system	ft. or in.
b_z	Camera body z-coordinate in world coordinate system	ft. or in.
d	Abstract distance between near image plane and far image plane	ft. or in.
fov_x	Camera field-of-view angle in the image x-direction	deg. or rad.
fov_y	Camera field-of-view angle in the image y-direction	deg. or rad.
G	Gaussian operator	Grayscale matrix
h	Altitude	ft. or in.
h	Image height	pixels
H	Hough transform	Grayscale matrix
H	Kalman observation matrix	6x7 matrix
I	Image	RGB matrix
J	Jacobian matrix of NDC to state variable derivatives	8x6 matrix
K	Optimal Kalman gain matrix	6x6 matrix
l	Runway length	ft. or in.
M	Combined Affine Transformation matrix	4x4 matrix
p_{2D}	Runway corner position vector in 2D image coordinates	pixels
p_{3D}	Runway corner position vector in 3D world coordinates	ft.
p_{ndc}	Runway corner position vector in 2D normalized device coordinates	-
P	Perspective matrix	4x4 matrix

Symbol	**Description**	**Units**
P	Covariance of optimal state estimation uncertainty	-
Q	Process noise covariance	7x7 matrix
R	Measurement noise covariance	6x6 matrix
R_x	Rotation matrix about x-axis	4x4 matrix
R_y	Rotation matrix about y-axis	4x4 matrix
R_Φ	Roll angle rotation matrix	4x4 matrix
R_Θ	Pitch angle rotation matrix	4x4 matrix
R_Ψ	Heading angle rotation matrix	4x4 matrix
T	Translation matrix	4x4 matrix
U	Total vehicle velocity	ft./sec. or in./sec.
u_Φ	Roll rate control input	rad./sec. or deg./sec.
u_Θ	Pitch rate control input	rad./sec. or deg./sec.
u_Ψ	Yaw rate control input	rad./sec. or deg./sec.
w	Image width	pixels
w	Runway width	ft. or in.
w_{clip}	Clipping w-coordinate	-
$\hat{x}$	Optimal state estimate	7x1 matrix
x_{clip}	Clipping x-coordinate	-
x_{im}/x_{img}	Image x-coordinate	pixels
x_{ndc}	Normalized device x-coordinate	-
x_{obj}	Object x-coordinate in world coordinate system	ft. or in.
y_{clip}	Clipping y-coordinate	-

Symbol	Description	Units
y_{im}/y_{img}	Image y-coordinate	pixels
y_{ndc}	Normalized device y-coordinate	-
y_{obj}	Object y-coordinate in world coordinate system	ft. or in.
z	State observation	6x1 matrix
z_{clip}	Clipping z-coordinate	-
z_{ndc}	Normalized device z-coordinate	-
z_{obj}	Object z-coordinate in world coordinate system	ft. or in.

Greek Symbols

Symbol	Description	Units
δb_x	Error in body x-position	in. or ft.
δb_y	Error in body y-position	in. or ft.
δb_z	Error in body z-position	in. or ft.
$\delta\Theta$	Error in pitch angle	deg. or rad.
$\delta\Phi$	Error in roll angle	deg. or rad.
$\delta\Psi$	Error in yaw angle	deg. or rad.
Θ	Pitch angle	deg. or rad.
κ	Adaptive scaling factor for Newton's method	-
υ	Frequency in Fourier transform corresponding to y-dimension	rad./sec.
ρ	Radius in Hough transform	pixels
σ_x	Standard deviation of Gaussian smoother in x-direction	pixels
σ_x	Standard deviation of Gaussian smoother in y-direction	pixels

Symbol	**Description**	**Units**
φ	Angular orientation in Hough transform	deg. or rad.
Φ	Roll angle	deg. or rad.
Ψ	Heading angle	deg. or rad.
Ω	Frequency in Fourier transform corresponding to x-dimension	rad./sec.

Acronyms

Symbol	**Description**
6DOF	Six Degree-of-Freedom
CPU	Central Processing Unit
EKF	Extended Kalman Filter
FOV	Field-of-View
FPGA	Field-Programmable Gate Array
GPS	Global Positioning System
NDC	Normalized Device Coordinate
RMS	Root-Mean-Square
SSE	Sum-Square-Error
SVD	Singular Value Decomposition
UAV	Unmanned Aerial Vehicle
VFR	Visual Flight Rules

List of Figures

Page #

List of Tables

1 Introduction

1.1 Overview and Motivation

This work is a further development of a project started in November of 2010 for EECS 741 at the University of Kansas to develop a vision-based state estimation algorithm for UAV autolanding and other autorecovery systems. Current state estimation methods for 6DOF mobile robots typically require the use of GPS. A recent government report [1] identified GPS as a national security threat due to its susceptibility to jamming. Devices capable of such denial-of-service jamming attacks are available for as little as 30 2011 US Dollars [2]. Alternative methods for UAV navigation must therefore be developed, particularly for military UAVs operating in hostile environments. In addition to the denial-of-service problem, most commonly-used GPS units do not meet performance requirements for fine-grained navigation tasks such as UAV landing on runways, on helipads, or in nets. This Thesis demonstrates a proof-of-concept of a method that can be modified to meet these requirements in VFR conditions and is more difficult to jam than GPS.

Some sources of GPS service unavailability are dense cloud cover, solar flares, and permanent obstructions such as trees and buildings. While any of these could certainly pose a problem to robot navigation, a more serious threat is presented by hostile action. GPS is now a staple of modern life: it is used in everything from navigation of automobiles to timestamping financial transactions to controlling ships in harbor. It is possible that terrorists or rogue states could identify the United States' or its allies' GPS infrastructure as the lynchpin in transportation, military, and even financial infrastructures. A recent report from the National

Space-Based Positioning, Navigation, and Timing Advisory Board identified GPS as a national security threat for these reasons, stating [1]:

> The United States is now critically dependent on GPS. For example, cell phone towers, power grid synchronization, new aircraft landing systems, and the future FAA Air Traffic Control System (NEXGEN) cannot function without it. Yet we find increasing incidents of deliberate or inadvertent interference that render GPS inoperable for critical infrastructure operations.

Another recent report in *New Scientist* [2] described a several-hour disruption in air-traffic control, emergency pagers for doctors, sea-traffic control, cell phones, and ATMs in San Diego due to a GPS-jamming exercise performed by the Navy. As dangerous as such attacks are, a potentially more damaging scenario could be caused by GPS-spoofing devices: signals that would drown-out real GPS signals and fool receivers into thinking they were only slightly offset from their true position. Such devices would have catastrophic effects on UAV autolanders, and could be extremely difficult to detect since GPS receivers would have no indication of a failure.

GPS is very good for medium-resolution navigation requirements with position accuracy to better than a 30 foot radius with 95% probability [3], [4]. More advanced units with RTK functionality provide accuracy on the order of several inches and even better precision in perfect conditions. However, the performance of such units is subject to the availability of satellites, dynamic conditions, and even geographical features. The accuracy provided by standard GPS receivers is not sufficient for fine-grain navigation tasks. RTK-enabled GPS units certainly do provide the required performance for fine-grained navigation; however, all units suffer equally from problems introduced by environmental conditions.

Vision-based state estimation for robotic mobile platforms is an appealing method due to its passive nature and its inherent ability to produce results based on the physical surroundings of the platform. The radiant intensity of common lighting scenarios is also much higher than the intensity of electromagnetic waves used in other positioning devices (radar, lidar, magnetometers, etc.), making optical sensing much more robust against jamming techniques. Cameras also generally reject radiation from outside their field of view, unlike ommidirectional GPS antennas that can be jammed from any aspect. Digital cameras are ubiquitous, inexpensive, lightweight, low power, and not nearly as susceptible to electromagnetic interference as the previously-mentioned methods. With powerful embedded processors needed to drive computationally-intensive vision algorithms becoming available in ever-smaller and lower-power packages, vision-based navigation methods are now viable on all but the smallest class of robotic platforms.

1.2 Previous Work

The method developed in this Thesis for robot state estimation is tailored for autonomous landing of unmanned aerial vehicles (UAVs), but is directly applicable to any class of robot when landmarks of known geometry are available in the operational environment. Several successful attempts have been made to develop very specific algorithms, primarily for rotorcraft [5], [6], [7], [8]. However, these algorithms all rely on a coplanar assumption and a customized landing pad, and are not necessarily suitable for use with a fixed-wing vehicle. An effort was made to develop an autoland system for a fixed-wing UAV [9], but not all state variables were estimated. There are some other related localization systems that have been developed [10], but none as general as the system presented in this Thesis.

The approach used draws heavily from the overall method from Sharp et al [5]. This team designed a highly-tailored landing pad to enable full state estimation for a helicopter, shown in Figure 1.1. While they did achieve RMS accuracy to within 2 inches in all axes, the RMS error in Euler angles was 4.5 degrees in the worst axis to 1 degree in the best axis. It should be noted that their system was successfully developed, tested, and used in an autonomous helicopter landing.

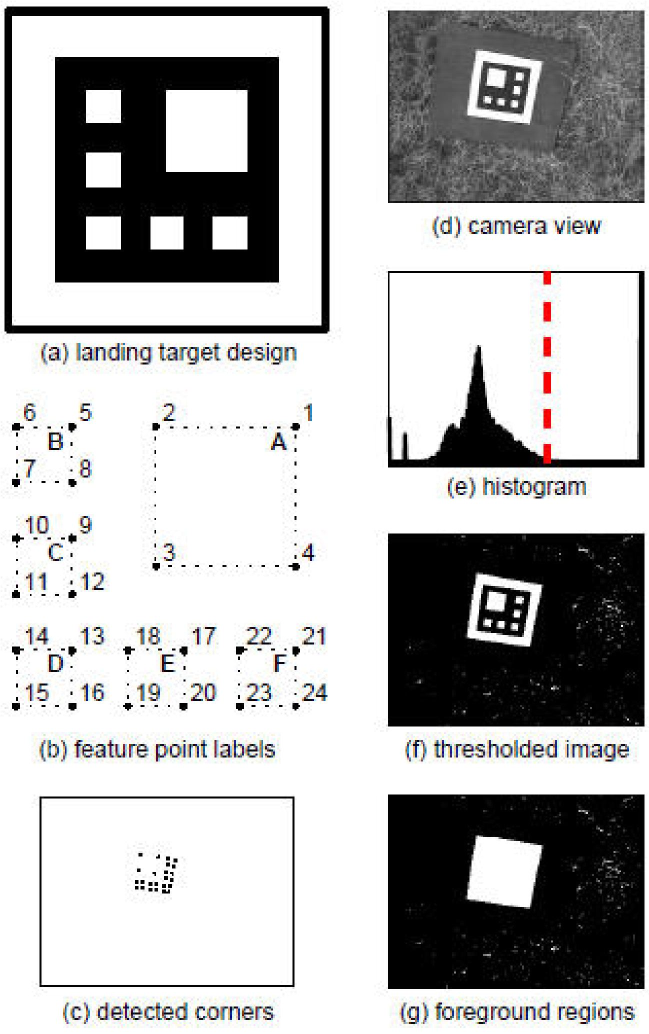
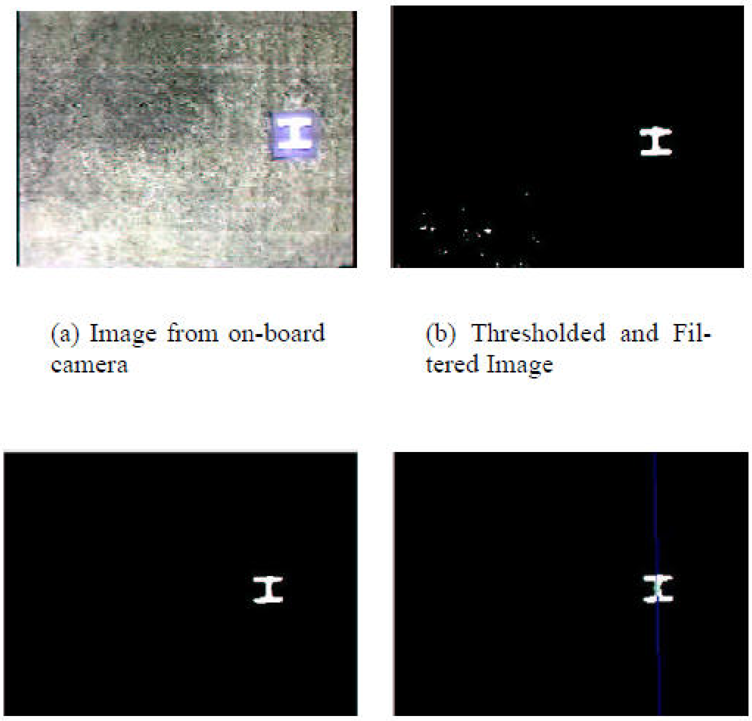


Figure 1.1: Landing Target Design from Sharp et al [5]

Saripalli et al took a similar approach, and were able to achieve a mean error in orientation of six degrees and a mean error in position of just over a foot using a 4 foot by 4 foot helipad. Figure 1.2 shows their landing pad design.



(a) Image from on-board camera

(b) Thresholded and Filtered Image

(c) Segmented Image

(d) Final Image

Figure 1.2: Landing Pad Design from Saripalli et al [6]

Cesetti et al took a different approach and developed a system that uses natural landmarks and SIFT (Scale-Invariant Feature Transform) features in combination with satellite imagery to estimate position in all three axes as well as heading [11]. However, this approach only works if the image is taken normal to the ground plane (i.e., pitch and roll are both constant), which certainly cannot be guaranteed on an airplane during landing. It could be used, however, to provide a navigation solution during steady level flight if GPS is unavailable.

Frew et al employed SLAM (Simultaneous Localization and Mapping) in conjunction with an unscented Kalman Filter and adaptive receding horizon control to solve the problem of UAV navigation in forests and urban environments. While useful for these scenarios, this method is not suitable for the problem of fine-grained navigation and guidance all the way through recovery.

1.3 Proposed Solution

The objective of this research is to develop a vision-based full state-estimation algorithm suitable for use not only on UAVs, but any robotic or manned platform that is within line-of-sight of landmarks of known geometry. The algorithm developed could be used as a landing navigation solution for UAVs, optionally-piloted vehicles, or even for pilot training. The algorithm in this Thesis can be readily adapted for use with: an arbitrary number of landmarks in arbitrary locations, photometric stereo vision to augment the state estimates with direct measurements of depth, or even multiple cameras if one camera is insufficient to capture every necessary landmark.

The algorithm is designed to be used on existing recovery targets (runways, helipads, and nets) that have clearly visible edges. Unicolor circular markers are placed on each corner of the recovery target to clearly denote their centroids on the image. The algorithm requires the color of the edges, the color of the markers, the 3D position of each corner of the recovery target, and at least a very rough initial state estimate that would be obtained from another higher-level vision-based navigation system or some other positioning system. The algorithm presented in this work makes the following assumptions:

- The corners of the runway are always in view of the camera. This would be accomplished by the flight control system in conjunction with proper placement of the camera.
- The markers used to denote the corners of the runway are of relatively constant, known color. Ground crews would ensure that these markings are clearly visible.

- The lines used to denote the perimeter of the runway are of relatively constant, known color.
- There is negligible barrel distortion in the camera lens. Most modern digital corrections remove such distortion, and if not, simple mathematical techniques can be employed to remove them.
- There is no dust or debris in the camera lens.
- Lighting conditions are such that the runway is clearly illuminated. Strategically placed lights could easily be employed that would enable the use of this algorithm at dawn, dusk, or night, in cloudy conditions, and other unfavorable lighting conditions.
- The vehicle is already close enough to the runway to clearly identify the corners of the runway.

This algorithm is intended for use during the final approach leg of vehicle recovery. It is intended that the vehicle control system would switch from either a course-grain or medium-grain navigation solution to this system when the vehicle is roughly lined up with the runway, helipad, or net. The course-grain navigation solution would provide the initial state estimate. The navigation solution provided by this algorithm would be valid until the threshold of the runway leaves the field of view of the camera. Modifications are proposed in the Conclusions that would enable the algorithm to handle this case as well.

2 Theoretical Development of the Method

2.1 Overview

An overview of the algorithm is given in Figure 2.1. The focus of this Thesis is on the core algorithm, which consists of the following three stages:

1. **Runway Identification**: the algorithm for picking the optimal two-dimensional quadrilateral in a given image and correlating the four resulting 2D points in the target image to four 3D points in the world coordinate system.
2. **State Observation**: the algorithm for estimating Euler angles and local position from the 2D quadrilateral and corresponding 3D geometry.
3. **Extended Kalman Filter (EKF)**: the algorithm for obtaining a near-optimal estimate of the true state of the vehicle using the current state observation in conjunction with prior knowledge of the vehicle’s state and dynamics.

In order to implement this algorithm onboard a real aircraft, it would be desirable to at least control the field-of-view angle, or the zoom, of the camera. In a more advanced system, the camera could also be mounted on a gimbal which could orient the camera in the direction of the runway regardless of the orientation or position of the aircraft.

The algorithm has as its inputs the image of the recovery target, the corner marker color, the edge color, and the runway target geometry. The onboard camera would provide the image, and the runway selector would provide the other information. The state estimation would be passed to both the camera controller to adjust field-of-view angle as well as the autoland controller.

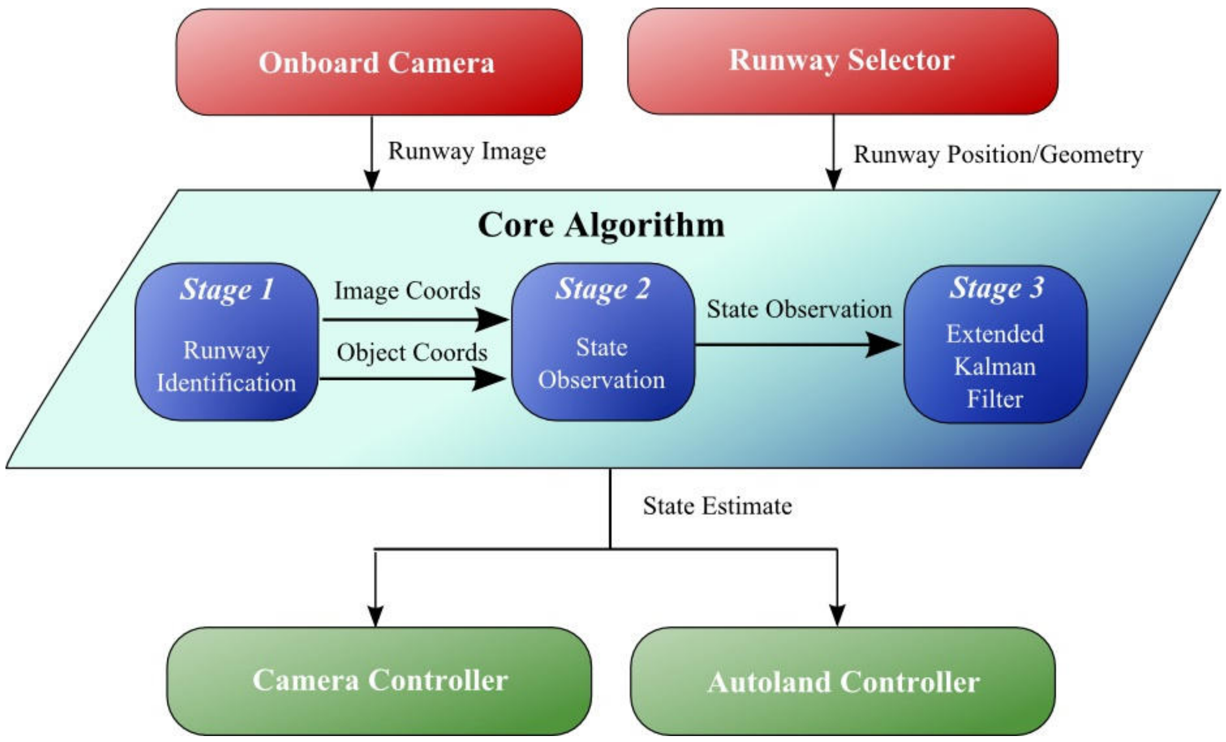


Figure 2.1: Algorithm Overview and Integration with Autoland System

2.2 *Coordinate Systems and Image Geometry*

Figure 2.2 shows the conventions used for the runway coordinate system.

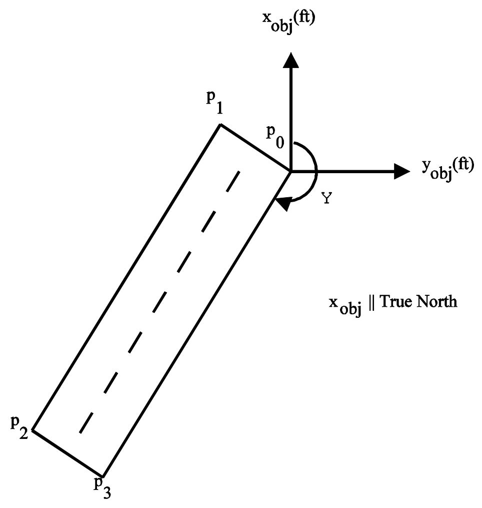


Figure 2.2: Local Coordinate System

Figure 2.3 shows the conventions used for the image geometry. The figure illustrates the conventions for the field-of-view angles in both directions, as well as so-called “normalized device coordinates” (NDCs) used to normalize each image on the interval [-1,1] in both the vertical and horizontal directions. The equations used for this normalization are Eq. 2.1 and 2.2.

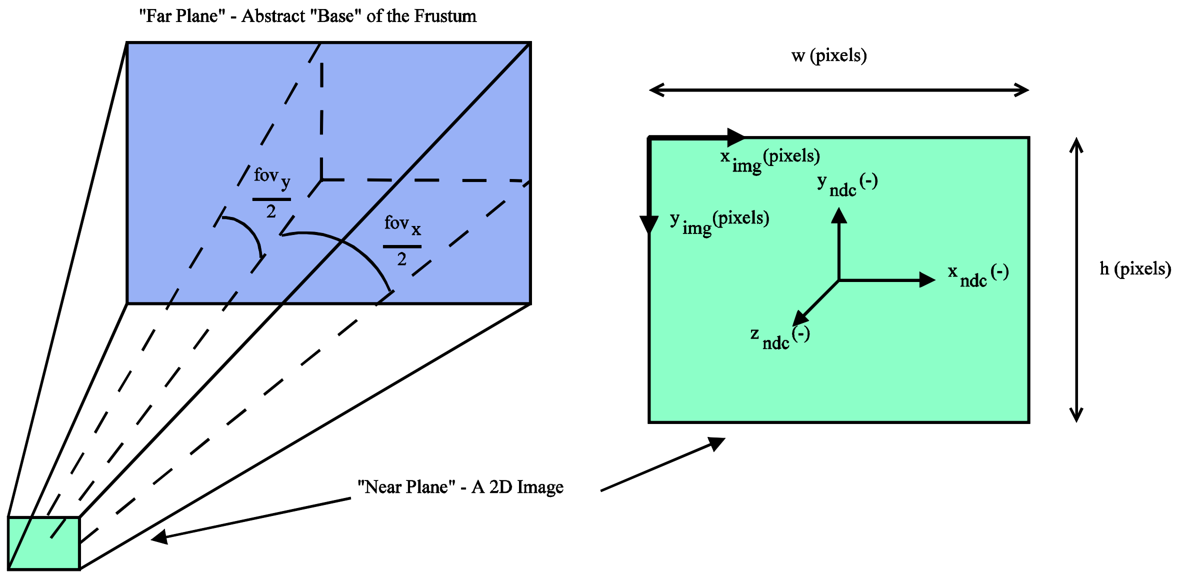


Figure 2.3: Image Geometry and Coordinate Systems

$$x_{ndc} = \left(x_{img} - {}^{w}\!/_{2}\right) / \left({}^{w}\!/_{2}\right) \qquad [2.1]$$

$$y_{ndc} = -\left(y_{img} - {}^{h}\!/_{2}\right) / \left({}^{h}\!/_{2}\right) \qquad [2.2]$$

If *d* is the distance separating the near plane and the far plane, then the field of view angles in the x- and y-directions are related as shown in Eqs. 2.3 and 2.4.

$$fov_x = 2\tan^{-1}\left(\frac{w}{2d}\right) \qquad [2.3]$$

$$fov_y = 2\tan^{-1}\left(\frac{h}{2d}\right) \qquad [2.4]$$

Thus, the field-of-view in the *y*-direction can be solved for in terms of the field-of-view in the *x*-direction and the image aspect ratio as shown in Eq. 2.5. This is helpful for converting between the convention used in this Thesis and the convention used in FlightGear.

$$fov_y = 2\tan^{-1}\left(\frac{\tan\left(\frac{fov_x}{2}\right)}{\left(\frac{w}{h}\right)}\right) = 2\tan^{-1}\left(\frac{\tan\left(\frac{fov_x}{2}\right)}{AR}\right) \qquad [2.5]$$

2.3 Runway Identification

2.3.1 Image Contrasting Algorithm

The first stage of the algorithm calculates a high-contrast image from the target image based on a color of interest. Two SSE images are generated: one SSE from the target marker color and one SSE from the target edge color.

This is a simple technique that generally greatly reduces the subspace of problems that must be solved by later stages of the algorithm. The algorithm simply computes the sum-square error in image *I* from a target color defined by <*r, g, b*> as shown in Eq. 2.6.

$$I_{SSE_{i,j}} = \left(I_{R_{i,j}} - r\right)^2 + \left(I_{G_{i,j}} - g\right)^2 + \left(I_{B_{i,j}} - b\right)^2 \qquad [2.6]$$

Figure 2.1 shows the square-root (for visualization purposes) of the Sum-Square Error from the target color in a sample image.

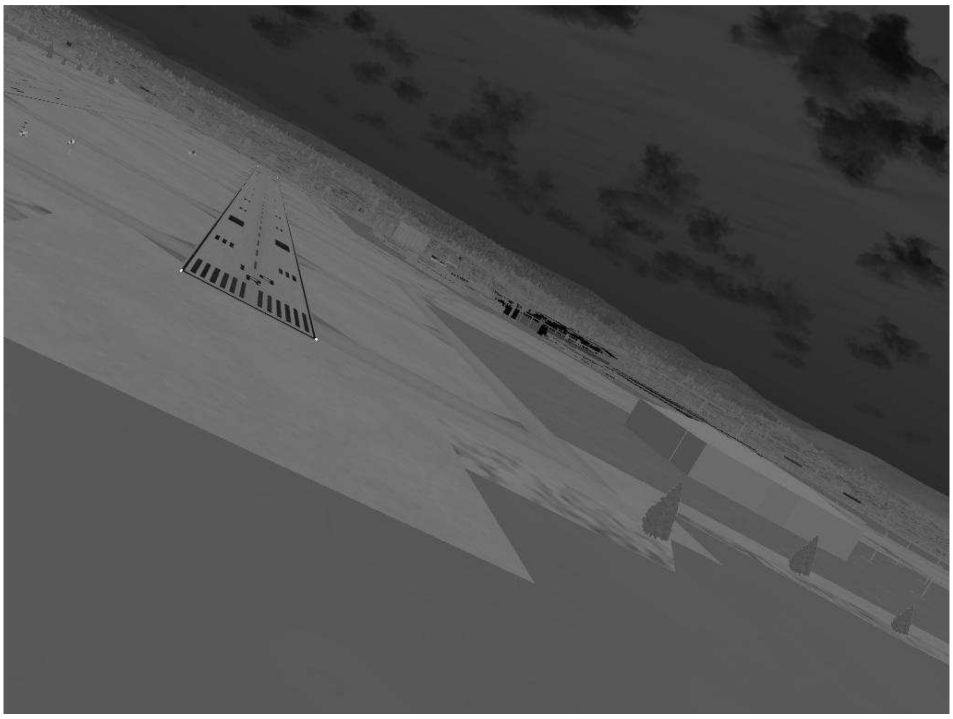


Figure 2.4: √(SSE) from Marker Color

2.3.2 Clustering Algorithm

The purpose of the clustering algorithm is to determine which pixels in a target image belong to contiguous marker candidates and return the center coordinates of each resulting cluster. A subset of these clusters will ultimately constitute the 2D screen coordinates that are mapped to 3D world coordinates and passed to the state estimator. The clustering algorithm has the following inputs: SSE from the target marker color; threshold value; search radius for clustering. It outputs a set of clusters with x and y image-space coordinates.

The clustering algorithm first thresholds the SSE image. It then iterates through each pixel that passes the threshold value. If the pixel is within proximity of a cluster (i.e., within the search radius of any other pixel in the cluster), it is added to the cluster. If it does not belong to an existing cluster, a new cluster is created with the pixel as the only member. In both cases, the pixel is removed from the thresholded set. This process is repeated until no pixels remain in the thresholded set. Each cluster's mean is calculated during this process.

Figure 2.5 shows an image with 7 pixel clusters (dilated for emphasis). This is the output of the algorithm with Figure 2.4 as the input image.

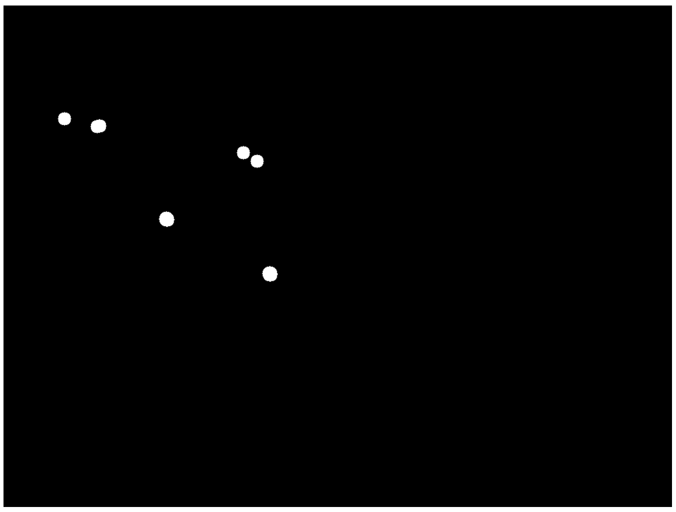


Figure 2.5: An Example Image with 7 Clusters

The tabulated output of the algorithm is shown in Table 2.1.

Table 2.1: Clustering Algorithm Output

Cluster #	x	y
1	94	174
2	144	186
3	147.5	185
4	250.46	327.82
5	368	226
6	389	239
7	408.75	411.58

2.3.3 Canny Edge Detector

The second stage of the algorithm uses a Canny Edge Detector [13] to locate the edges in the image. The Canny Edge detector is a commonly-used digital image processing algorithm that relies on the gradient magnitude and the gradient direction of an image to locate edges. A

Gaussian is used in combination with the discrete derivative to reduce noise in the resulting edge map. The inputs to the detector are the standard deviation value to use for the Gaussian operator, the minimum threshold value to use for line strength, and the image itself. The 2D Gaussian is given by Eq. 2.7.

$$G(x,y) = e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)} \qquad [2.7]$$

Taking the partial derivatives with respect to *x* and *y* gives Eqs. 2.8 and 2.9.

$$\frac{\partial G(x,y)}{\partial x} = -\frac{x}{\sigma_x^2} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)} \qquad [2.8]$$

$$\frac{\partial G(x,y)}{\partial y} = -\frac{y}{\sigma_y^2} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)} \qquad [2.9]$$

It is desired to convolve the image *I* with the images generated by the two partial derivatives of *G* to generate the image gradient in the *x*-direction and the *y*-direction. Rather than performing the convolution directly, the Convolution Theorem is used. Direct computation is feasible when σ is small; but it is generally much faster to compute the convolution using Fast Fourier Transforms. The convolutions are given by Eqs. 2.10 and 2.11.

$$\left(I * G_x\right)(x,y) = \mathcal{F}^{-1}\left[\mathcal{F}[I(x,y)](\omega,\upsilon)\cdot\mathcal{F}[G_x(x,y)](\omega,\upsilon)\right](x,y) \qquad [2.10]$$

$$\left(I * G_y\right)(x,y) = \mathcal{F}^{-1}\left[\mathcal{F}[I(x,y)](\omega,\upsilon)\cdot\mathcal{F}[G_y(x,y)](\omega,\upsilon)\right](x,y) \qquad [2.11]$$

Taking the Fourier Transform of the 2D Gaussian derivatives in the *x*- and *y*- directions give Eqs. 2.12 and 2.13.

$$\mathcal{F}_x\left[\frac{\partial G(x,y)}{\partial x}\right](\omega) = -i\sigma_x \omega e^{\left(-\omega^2\frac{\sigma_x^2}{2}-\frac{y^2}{2\sigma_y^2}\right)} \qquad [2.12]$$

$$\mathcal{F}_{x,y}\left[\frac{\partial G(x,y)}{\partial x}\right](\omega,\upsilon)=-i\sigma_x\sigma_y\omega e^{\left(-\omega^2\frac{\sigma_x^2}{2}-\upsilon^2\frac{\sigma_y^2}{2}\right)} \quad [2.13]$$

By symmetry, the same process gives Eq. 2.14 for the *y*-direction.

$$\mathcal{F}_{y,x}\left[\frac{\partial G(x,y)}{\partial y}\right](\upsilon,\omega)=-i\sigma_x\sigma_y\upsilon e^{\left(-\omega^2\frac{\sigma_x^2}{2}-\upsilon^2\frac{\sigma_y^2}{2}\right)} \quad [2.14]$$

The gradient operators from Eqs. 2.13 and 2.14 are shown in Fourier space in Figure 2.6.

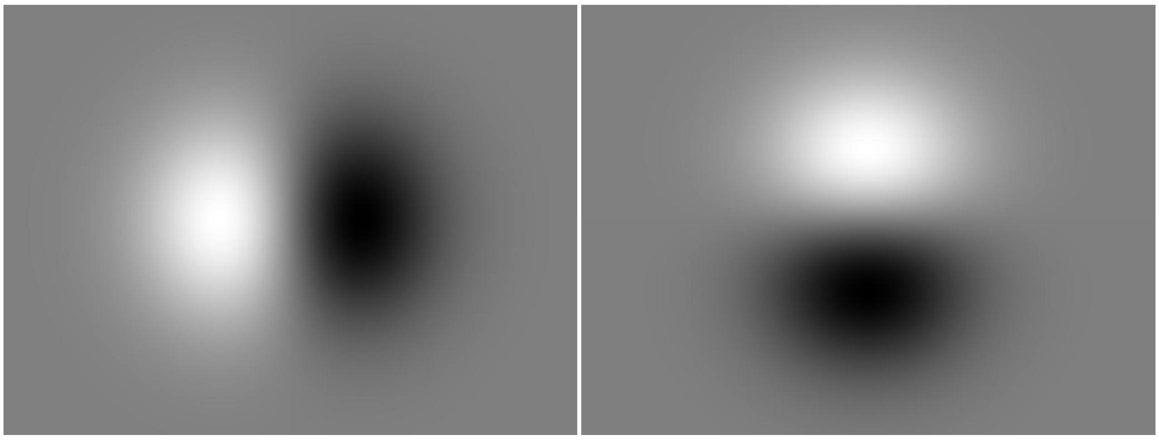


Figure 2.6: imag(G_x) (Left) and imag(G_y) (Right) with σ = 1 px

The overall gradient magnitude and gradient direction are given by Eqs. 2.15 and 2.16. The results of the operations are shown in Figure 2.7 for a sample case.

$$I_{GM}(x,y)=\sqrt{\left((I*G_x)(x,y)\right)^2+\left((I*G_y)(x,y)\right)^2} \quad [2.15]$$

$$I_{GD}(x,y)=\operatorname{atan2}\left((I*G_y)(x,y),(I*G_x)(x,y)\right) \quad [2.16]$$

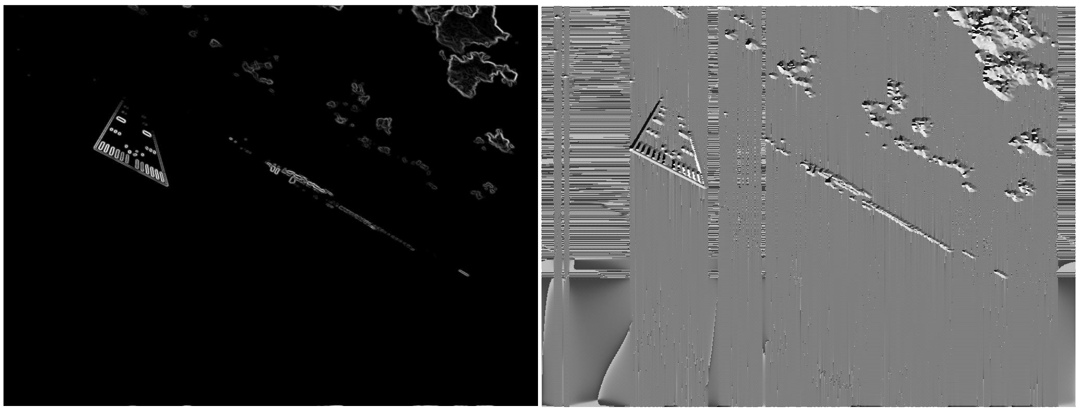


Figure 2.7: Gradient Magnitude and Direction with σ = 1 of a Sample Image

Having finally computed the Gradient Magnitude and Direction, the Canny Edge detector performs the following steps:

1. Threshold the Gradient Magnitude to generate a binary image of sufficiently strong edge candidates. A constant value of 10% overall strength is used in the algorithm.
2. Discretize the edge directions into 45 degree bins.
3. For each pixel in the image from step 1, if the pixel lies in a region with a consistent direction, then it is considered to be an edge.

The results of the Canny operator on a sample image are shown in Figure 2.8.

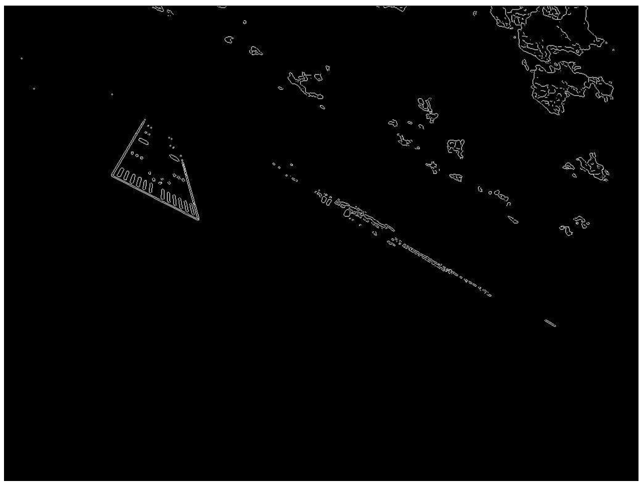


Figure 2.8: Results of Canny Edge Detector with σ = 1 and t = 10% on a Sample Image

2.3.4 Hough Transform

A Hough transformation [14] is then performed on the output of the Canny Edge detector. This variant of the Hough transform takes in a binary image and returns an image that indicates where the strongest (or alternatively, most highly-correlated) lines in the image lie. This process works by considering every point in image space to be a sinusoid in Hough space. Regions with many intersections of sinusoids in Hough space correspond to strongly-correlated lines in image space. The mapping of points in image space to sinusoids in Hough space is given by Eq. 2.17.

$$\rho = x\cos\phi + y\sin\phi \qquad [2.17]$$

The algorithm works in two steps:

1. Initialize the Hough map to all zeros
2. Whenever a non-zero element is encountered in Image space, increment the values along the corresponding sinusoid in the Hough map.

The Hough Transform of Figure 2.8 is shown in Figure 2.9. The most strongly-correlated lines are located at the peaks of the Hough transform.

Figure 2.9: Hough Transform of a Test Image

2.3.5 Cluster-Line Proximity Filter

To reduce the number of marker candidates in an image, a filter is employed which checks the proximity of a marker to a strong edge. Given marker candidate A, in order to pass the filter there must exist a marker B such that the line formed by AB is essentially collinear with a runway edge. The proximity filter has the following inputs: a set of input candidate markers, a Hough transform, the range of $\phi$and ρvalues in the Hough transform, minimum line strength on the normalized interval [0,1], and the search radius in pixels.

The line formed by the two candidate markers in image-space can be found in Hough-space by finding the solution to the following system of equations:

$$\begin{aligned} x_1 \cos(\phi) + y_1 \sin(\phi) = \rho \\ x_2 \cos(\phi) + y_2 \sin(\phi) = \rho \end{aligned}$$

This can be rewritten as:

$$\begin{bmatrix} x_1 & -1 \\ x_2 & -1 \end{bmatrix} \begin{bmatrix} \cot\phi \\ \dfrac{\rho}{\sin\phi} \end{bmatrix} = \begin{bmatrix} -y_1 \\ -y_2 \end{bmatrix} \Leftrightarrow AB = C$$

And then the solution is given as:

$$\begin{aligned} B &= A^{-1}C \\ \phi &= \cot^{-1}(B_{1,1}) \\ \rho &= B_{2,1} \sin(\phi) \end{aligned}$$

Table 2.2 shows an example input set of clusters for the Cluster-Line proximity filter. Figure 2.10 shows the thresholded Hough transform with an intensity of 40% on the left and the Hough transform with all $\binom{8}{2} = (\frac{8!}{2!(8-2)!}) = 28$ possible line searches within a search radius of 5 pixels on the right. Note that some of the search spaces are quite close and so may not be clearly distinguishable.

Table 2.2: Cluster-Line Proximity Filter Input

Cluster (#)	x (pixels)	y (pixels)
1	58	417
2	63	418
3	144	396
4	149.2	390.6
5	297.79	594.29
6	513	314
7	538	314
8	782.84	596.15

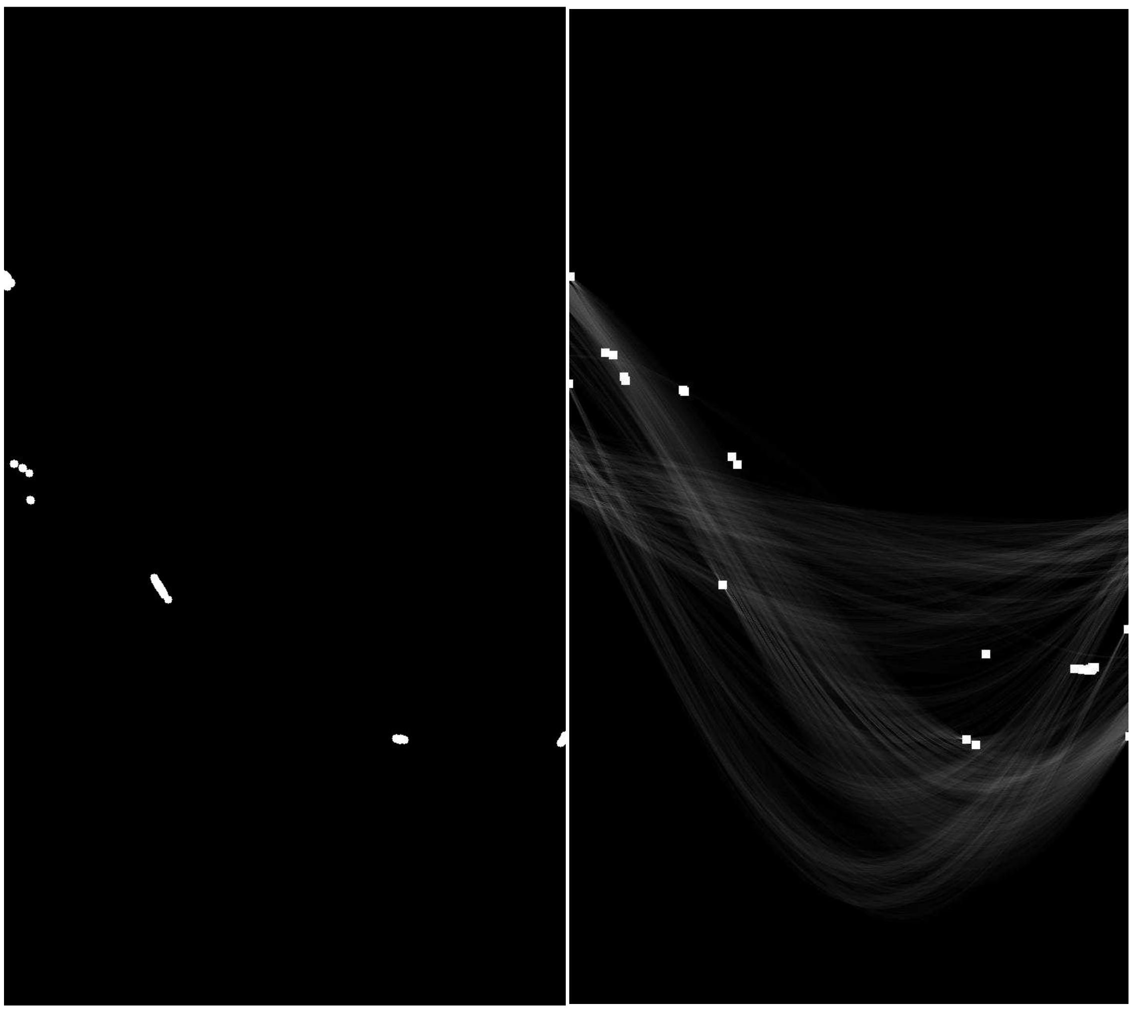


Figure 2.10: Threshold Mask (Left) and Searched (Right) Hough Space Images

Table 2.3 shows the resulting filtered clusters using a threshold value of 40% intensity.

Table 2.3: Filtered Clusters

Cluster (#)	x (pixels)	y (pixels)
5	297.79	594.29
6	513	314
7	538	314
8	782.84	596.15

2.3.6 2D Image Coordinate to 3D World Coordinate Optimization

The final step in the algorithm before state observation is performed is to match the clusters from the previous step with real 3D world coordinates. This step relies on prior knowledge to perform the optimization. Given the previous map $\{\vec{p}_{2D_{k-1}} \rightarrow \vec{p}_{3D_{k-1}}\}$, the new optimal map minimizes the sum-square distance between the previous set of 2D points and the new set of 2D points while maintaining the same ordering as shown in Eq. 2.18.

$$\left\{\vec{p}_{2D_k} \rightarrow \vec{p}_{3D}\right\} = \left\{\underset{\vec{p}_{2D_k}}{\arg\min} \sum_{i=1}^{N} \left(p_{2D_{k_i}} - p_{2D_{(k-1)_i}}\right)^2 \rightarrow \vec{p}_{3D}\right\} \quad [2.18]$$

Therefore if *n* is the number of candidate clusters and *k* is the number of desired clusters, then the number of permutations necessary to find the optimal result is $\binom{n}{k}$. Since this optimization runs in *O(n!)* time, it is clear that the previous steps vastly diminish the subspace of problems that must be solved by this stage of the algorithm.

2.4 State Observation Algorithm

The state observation algorithm is responsible for taking an initial estimate of the state, the camera's aspect ratio and field-of-view angle, and a list of normalized device coordinates along with these points' corresponding physical coordinates and obtaining an optimal estimate of the observer's state. In summary, the constants in the algorithm are:

- AR, the aspect ratio of the camera

- fovy, the field-of-view angle of the camera

The dynamic inputs are:

- A mapping of 2D normalized device coordinates to their corresponding global real-world 3D coordinates.
- An initial estimate of the observer's state (or in the case of a moving observer, an *a priori* estimate of the new observer state).

The outputs of the algorithm are then as follows:

- Three Euler angles defining the observer's orientation.
- A 3D displacement vector from the origin in the image.

It should here be noted that a few things are ignored in this algorithm, including primarily spherical distortion and asymmetry in the lens. However, many modern digital cameras are pre-calibrated to remove the effects of spherical distortion and this has so far not proven to be an issue.

The first step in developing this algorithm is to formulate a set of equations that will project a 3D point onto a 2D image using Euler angles and a translation vector. The method used for this is based on a combination of conventions in aerospace engineering [15] and the method of projection used in OpenGL [16]. In order to allow projections and translations, Affine Transforms are used throughout the algorithm.

The camera is assumed to be mounted at the C.G. of the aircraft, orthogonally situated with the *z*-axis pointing ahead of the aircraft and the *x*-axis pointing in the starboard spanwise direction. While this is not realistic in a typical installation, additional rotation or translation matrices could easily be used to account for the differences between the aircraft C.G. and the camera location. Since the transformation matrices used assume that the *x*-direction is initially pointed right and the *y*-direction is pointed up, the transformation matrix must be initialized to account for the necessary rotation. The necessary rotation is performed with 90 degree rotations about the *y*- and *x*-axes as shown in Eqs. 2.19 and 2.20.

$$R_y = \begin{bmatrix} \cos\left(\frac{\pi}{2}\right) & 0 & \sin\left(\frac{\pi}{2}\right) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\left(\frac{\pi}{2}\right) & 0 & \cos\left(\frac{\pi}{2}\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.19]$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right) & 0 \\ 0 & \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.20]$$

The Euler rotation matrices are standard for aerospace coordinate systems, and are shown in Eqs. 2.21, 2.22, and 2.23.

$$R_\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) & 0 \\ 0 & \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.21]$$

$$R_\Theta = \begin{bmatrix} \cos(\Theta) & 0 & \sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Theta) & 0 & \cos(\Theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.22]$$

$$R_\Psi = \begin{bmatrix} \cos(\Psi) & -\sin(\Psi) & 0 & 0 \\ \sin(\Psi) & \cos(\Psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.23]$$

The translation matrix moves the camera to the origin, and is given in Eq. 2.24.

$$T = \begin{bmatrix} 1 & 0 & 0 & b_x \\ 0 & 1 & 0 & b_y \\ 0 & 0 & 1 & b_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.24]$$

Finally, the projection matrix is given in Eq. 2.25. Note that the *znear* and *zfar* parameters essentially drop out of the equation since normalized z-coordinates are not available, but are nonetheless shown for the sake of completeness. Normalized z-coordinates on the interval [-1, 1] represent geometry inside the viewing volume. Since no depth information is available and the z-coordinates are not used, this information is not strictly necessary for the projection.

$$P = \begin{bmatrix} \cot\left(\frac{fov_y}{2}\right) \cdot \frac{1}{AR} & 0 & 0 & 0 \\ 0 & \cot\left(\frac{fov_y}{2}\right) & 0 & 0 \\ 0 & 0 & \frac{znear + zfar}{znear - zfar} & \frac{2znear \cdot zfar}{znear - zfar} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad [2.25]$$

Combining all of these matrices in the proper order yields Eq. 2.26.

$$M = P \times R_y \times R_x \times R_\Phi \times R_\Theta \times R_\Psi \times T \qquad [2.26]$$

With this transformation matrix available, it is then possible to transform object coordinates – that is, 3D runway coordinates in the world coordinate system – into clipping coordinates as shown in Eq. 2.27.

$$\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix} = M \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ 1 \end{pmatrix} \qquad [2.27]$$

Finally, Eq. 2.28 shows the transformation of the clipping coordinates into normalized device coordinates by a projection operation. This is the final step in the forward-projection of object coordinates onto the viewing plane.

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \end{pmatrix} \Big/ w_{clip} \qquad [2.28]$$

This method is well-known and used in many modern rendering systems, with some slight variations. However, the challenge in this algorithm is to invert this process, and solve for the state variables given normalized device coordinates and object coordinates.

The resulting system of non-linear equations may be solved using Newton's Method with an adaptive step size. While the resulting equations are certainly extremely non-linear, they are locally quite linear in general, making this method a reasonable choice. In order to use Newton's Method, the Jacobian matrix must first be calculated as shown in Eq. 2.29.

$$J = \begin{bmatrix} \frac{\partial x_{1_{ndc}}}{\partial \Phi} & \frac{\partial x_{1_{ndc}}}{\partial \Theta} & \cdots & \frac{\partial x_{1_{ndc}}}{\partial b_z} \\ \frac{\partial y_{1_{ndc}}}{\partial \Phi} & \frac{\partial y_{1_{ndc}}}{\partial \Theta} & \cdots & \frac{\partial y_{1_{ndc}}}{\partial b_z} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{n_{ndc}}}{\partial \Phi} & \frac{\partial y_{n_{ndc}}}{\partial \Theta} & \cdots & \frac{\partial y_{n_{ndc}}}{\partial b_z} \end{bmatrix} \qquad [2.29]$$

Each of the 12 unique terms in the Jacobian matrix was analytically calculated using GNU/Maxima and imported directly in closed-form into MATLAB. With the Jacobian calculated, it is then possible to apply Eq. 2.30 to converge to a solution. Here, $\left(\vec{\tilde{p}}_{ndc} - \vec{\hat{p}}_{ndc_k}\right)$ is the difference between the observed normalized device coordinates in the measured image and the estimate of the projection of those coordinates based on the current transformation matrix M_k and the corresponding world coordinates of the point. This equation is iterated until convergence is reached to a reasonable tolerance – typically in 5 to 10 iterations for the first observation and within 2-3 iterations for subsequent observations with better initial conditions.

$$\begin{pmatrix} \Phi \\ \Theta \\ \Psi \\ b_x \\ b_y \\ b_z \end{pmatrix}_{k+1} = \begin{pmatrix} \Phi \\ \Theta \\ \Psi \\ b_x \\ b_y \\ b_z \end{pmatrix}_{k} + \kappa J^{-1}\left(\vec{\tilde{p}}_{ndc} - \vec{\hat{p}}_{ndc_k} \right) \qquad [2.30]$$

The choice for κ – the adaptive step constant – is based on an assumption of the approximate local linearity of the system of equations. It is assumed that the transformation equation is roughly linear within $\pi/6$ radians of change in any Euler angle and within any change in distance. This encourages the solver to look for solutions that are close to the initial condition and improves the stability of the solver overall. The expression for κ is given in Eq. 2.31.

$$\kappa = \begin{cases} 1 & if \ \left|\max\left(abs\left|\{\Phi,\Theta,\Psi\}\right|\right)\right| < \pi/6 \\ 1/\max\left(abs\left|\{\Phi,\Theta,\Psi\}\right|\right) & otherwise \end{cases} \qquad [2.31]$$

It is clear from this process that exactly three two-dimensional points are required to solve the system. However, Newton’s Method may be generalized to overconstrained systems if the Moore-Penrose Matrix Pseudoinverse is utilized. Such a technique will yield a solution that is optimal in the Least-Squares sense, thus improving both the accuracy of the solution as well as convergence characteristics of the algorithm’s implementation. The pseudoinverse can be calculated using the Singular Value Decomposition of the Jacobian matrix [17]. If the SVD of J is given by Eq. 2.32, then the pseudoinverse of J is Eq. 2.33.

$$J = U\Sigma V^{*} \qquad [2.32]$$

$$J^{+} = V\Sigma^{+}U^{*} \qquad [2.33]$$

Since Σ is a diagonal matrix, Σ^{+} is given by taking the scalar reciprocal of each diagonal element of the matrix. This is the computational method used by MATLAB’s `pinv` function, which is utilized in this algorithm. Using this technique, the number of two-dimensional points that may be used for solving the system is on the interval $[3,\infty)$. If depth information were available, the number of permissible points would become $[2,\infty)$.

2.5 Extended Kalman Filter

2.5.1 System Dynamics

A very simplified set of vehicle dynamics (Eq. 2.34) has been contrived for simulating motion. The objective of this set of dynamics is to provide a simulation platform for mimicking the trajectory of an aircraft without becoming encumbered in the details of real aircraft dynamics. The added benefit of using a simplified set of dynamics is that it becomes easier to tune the Extended Kalman Filter (EKF).

The set of dynamics in question was developed using the following assumptions:

1. The vehicle always flies in the direction of the nose of the aircraft: i.e. no angle-of-attack or angle-of-sideslip.
2. The total forward velocity of the aircraft is constant.
3. Three control inputs directly and instantaneously affect the Euler angles.

In no way is this set of dynamics intended to be a substitute for real aircraft dynamics. However, the trajectory of an aircraft on approach can be mimicked with this set of dynamics and it is sufficient for demonstrating that this algorithm can track the trajectory of a moving body in flight.

Recalling that x_b, y_b, and z_b are the local coordinates of the vehicle with respect to p_0 of the runway, the system dynamics are given as shown in Eq. 2.34.

$$\begin{aligned}
\dot{b}_x &= U\cos\Theta\cos\Psi \\
\dot{b}_y &= U\cos\Theta\sin\Psi \\
\dot{b}_z &= -U\sin\Theta \\
\dot{\Phi} &= u_\Phi \\
\dot{\Theta} &= u_\Theta \\
\dot{\Psi} &= u_\Psi
\end{aligned} \qquad [2.34]$$

In non-linear state-space form, this can be written as shown in Eq. 2.35.

$$\begin{bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \\ \dot{b}_x \\ \dot{b}_y \\ \dot{b}_z \\ \dot{U} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cos\Theta\cos\Psi \\ 0 & 0 & 0 & 0 & 0 & 0 & \cos\Theta\sin\Psi \\ 0 & 0 & 0 & 0 & 0 & 0 & -\sin\Theta \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Phi \\ \Theta \\ \Psi \\ b_x \\ b_y \\ b_z \\ U \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_\Phi \\ u_\Theta \\ u_\Psi \end{bmatrix} \qquad [2.35]$$

For the purposes of simulation this model can be discretized into the form of Eq. 2.36 with a specified sampling time.

$$\vec{x}_{k+1} = \Phi_k \vec{x}_k + \Gamma_k \vec{u}_k \qquad [2.36]$$

The discretized version of the system dynamics is shown in Eq. 2.37.

$$\begin{bmatrix} \Phi \\ \Theta \\ \Psi \\ b_x \\ b_y \\ b_z \\ U \end{bmatrix}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \cos\Theta\cos\Psi\Delta T \\ 0 & 0 & 0 & 0 & 1 & 0 & \cos\Theta\sin\Psi\Delta T \\ 0 & 0 & 0 & 0 & 0 & 1 & -\sin\Theta\Delta T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Phi \\ \Theta \\ \Psi \\ b_x \\ b_y \\ b_z \\ U \end{bmatrix}_k + \begin{bmatrix} \Delta T & 0 & 0 \\ 0 & \Delta T & 0 \\ 0 & 0 & \Delta T \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_\Phi \\ u_\Theta \\ u_\Psi \end{bmatrix}_k \qquad [2.37]$$

2.5.2 Filter Design

A conventional linear Kalman Filter [18] consists of two steps: a prediction step and a correction step. The prediction step is performed using Eqs. 2.38 and 2.39 [19].

$$\hat{x}_{k+1}(-) = \Phi_k \hat{x}_k + \Gamma_k \vec{u}_k \qquad [2.38]$$

$$P_{k+1}(-) = \Phi_k P_k \Phi_k^T + Q_k \qquad [2.39]$$

The optimal Kalman gain K is calculated using Eq. 2.40. The correction step is then performed using Eqs. 2.40, 2.41, and 2.42 [19].

$$K_k = P_k(-) H_k^T \left[H_k P_k(-) H_k^T + R_k \right]^{-1} \qquad [2.40]$$

$$\hat{x}_{k+1}(+) = \hat{x}_{k+1}(-) + K_k \left[z_k - H_k \hat{x}_{k+1}(-) \right] \qquad [2.41]$$

$$P_{k+1}(+) = \left[P_k(-)^{-1} + H_k^T R_k^{-1} H_k \right]^{-1} \qquad [2.42]$$

The linear Kalman Filter may be extended to the non-linear case by linearizing about the current state and covariance. The non-linear multivariable state transition function and measurement equation are given by Eqs. 2.43 and 2.44.

$$\vec{x}_{k+1} = f\left(\vec{x}_k, \vec{u}_k\right) + \vec{w}_k \qquad [2.43]$$

$$\vec{z}_{k+1} = h\left(\vec{x}_k\right) + \vec{v}_k \qquad [2.44]$$

Eqs. 2.43 may be discretized by first forming the state-transition matrix as shown in Eq. 2.45.

$$\Phi_k = \left.\frac{\partial f}{\partial \vec{x}}\right|_{\hat{x}_k, \vec{u}_k} \qquad [2.45]$$

Similarly, the measurement matrix is formed as shown in Eq. 2.46.

$$H_k = \left.\frac{\partial h}{\partial \vec{x}}\right|_{\hat{x}_k} \qquad [2.46]$$

Since the outputs of the Stage 2 algorithm are simply Euler angles and Cartesian coordinates, there is no direct observation for U. Therefore the discrete measurement equation becomes Eq. 2.47.

$$\vec{z}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \vec{x}_k + \vec{v}_k \qquad [2.47]$$

Since this is a very non-linear system, it is difficult to analytically solve for *Q* (the process noise covariance) and *R* (the measurement noise covariance). For the purposes of this Thesis, *Q* and *R* are assumed to be the constant matrices shown in Eqs. 2.48 and 2.49. This says that the variance of the process noise is 1/10th of the measurement noise over an infinite period of time and that all noise is completely uncorrelated. This is perhaps not the optimal solution for the matrices, but it still yields good results that are shown in the next section.

$$Q = \begin{bmatrix} .01 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & .01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .01 \end{bmatrix} \qquad [2.48]$$

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad [2.49]$$

3 Algorithm Implementation

3.1 *System Structure*

3.1.1 Configuration Options

The system's configuration options are shown in Table 3.1. These options are intended to be static options that would be set prior to usage in a real-world scenario.

Table 3.1: Algorithm Configuration Options

Parameter Name	Parameter Description
`line_thresh`	Allowable SSE deviation from RGB line color.
`mrkr_thresh`	Allowable SSE deviation from RGB marker color.
`line_color`	Target RGB line color for runway edges.
`mrkr_color`	Target RGB marker color for runway markers.
`line_sigma`	Standard deviation for edge gradient Gaussian in pixels.
`mrkr_search_r`	Proximity search radius for marker in pixels.
`hough_rho_res`	ρ resolution for the Hough transform in pixels.
`hough_phi_range`	φ range for the Hough transform in degrees.
`hough_thresh`	Minimum line intensity for cluster proximity search.
`hough_search_r`	Maximum search radius for cluster proximity.
`AR`	Image aspect ratio.
`fovy`	Field-of-view angle in radians.
`znear`	Near z-coordinate (unused).
`zfar`	Far z-coordinate (unused).
`im_size`	Image size, [h w], in pixels

3.1.2 Algorithm Inputs

Table 3.2 shows the inputs to the algorithm. These are intended to be variables that could change from one iteration to another. Note that α is not intended to be a permanent part of the algorithm, but was introduced to quickly but crudely compensate for difficulties created by a natural image test.

Table 3.2: Algorithm Inputs

Input Name	Format	Description
img_raw	*h* x *w* x 3 RGB image	Raw RGB image
vision_cfg_props	Structured config data	Configuration properties
coord_map	4x4 matrix	Previous 2D Image to 3D world coordinate map
x_post	7x1 matrix	Previous optimal state estimate
P_post	7x7 matrix	Previous optimal covariance estimate
u_k	3x1 matrix	Current control input
alpha	Constant	Angle-of-attack estimate
dt	Constant	Sampling time interval

3.1.3 Algorithm Outputs

Table 3.3 shows the outputs of the algorithm. The optimal state estimate, optimal covariance estimate, and coordinate map should simply be passed into the algorithm on the next iteration, comprising a feedback loop.

Table 3.3: Algorithm Outputs

Output Name	Format	Description
x_post	7x1 matrix	New optimal state estimate
P_post	7x7 matrix	New optimal covariance estimate
z	6x1 matrix	Raw observation from the core algorithm
coord_map	4x4 matrix	New 2D Image to 3D world coordinate map
vision_perf	Structure	CPU load data for each stage in the algorithm
delta_ndc	4x12 matrix	Sensitivity to each $<x_{ndc},y_{ndc}>$ pair
delta_fovy	1x6 matrix	Sensitivity to fov_y

3.2 Complexity Analysis

Table 3.4 shows an algorithmic complexity analysis of each major sub-stage in the overall algorithm. This complexity analysis can be used with empirical data to project the effect of changes in the data size on the overall computational load on the target system. Most operations scale linearly, but there are a few exceptions:

- The Edge Detection algorithm is dominated by the Fast Fourier Transform used to apply the Gaussian smoothing operation, and this runs in linearithmic time.

- The Hough Transform runs in *O(n*m)*, although for practical purposes this can be approximated as *O*(n^2).
- The Cluster proximity algorithm is defined recursively and is quadratic in complexity.
- The mapping algorithm runs in factorial time, although the proximity algorithm is designed to keep the number of inputs to this algorithm very low.

Table 3.4: Algorithmic Complexity

Subsystem	Complexity	Meaning of n	Meaning of m
Image Acquisition	*O(n)*	Number of pixels in the image	-
Line SSE	*O(n)*	Number of pixels in the image	-
Marker SSE	*O(n)*	Number of pixels in the image	-
Edge Detect	*O(n log n)*	Number of pixels in the image	-
Hough Transform	*O(n*m)*	Number of pixels in the Hough Transform image	Number of detected edge pixels
Cluster	*O(n+m)*	Number of pixels in the image	Number of detected marker pixels
Cluster Proximity	*O*(n^2)	Number of total clusters detected	-
Cluster 2D to 3D Mapping	*O(n!)*	Number of candidate clusters	-
State Observation	*O(1)*	-	-
Kalman Filtering	*O(1)*	-	-

4 Results

4.1 Synthetic Tests with FlightGear

4.1.1 Simulation Setup

Several trajectories were set up using FlightGear v. 2.0.0 [20]. KSFO runway 19R was chosen as the target airport for performing tests on the algorithm since it provided an opportunity to demonstrate robustness in the presence of a cluttered scene with multiple runways.

Since FlightGear does not natively support the addition of items like markers, corner markers were artificially superimposed on each of the images. While this practice calls into question the results of the clustering algorithm, it enables simpler and more flexible testing of the state observation algorithm and Kalman filter in the presence of motion. Natural image tests are later performed to validate the clustering algorithm.

All sample images were rendered in FlightGear 2.0.0 on an Nvidia GeForce 9600GSO video card with the following rendering options:

- 1024x768 image resolution
- 55° field-of-view in the *x*-direction
- 8X Full-Screen Anti-Aliasing
- 16X Anisotropic Filtering

Table 4.1 shows the configuration options used for all of the synthetic tests.

Table 4.1: Configuration Options for Synthetic Tests with FlightGear

Name	Value
`line_thresh`	50^2
`mrkr_thresh`	50^2
`line_color`	[230 230 230]
`mrkr_color`	[255 0 0]
`line_sigma`	1
`mrkr_search_r`	10
`hough_rho_res`	2
`hough_phi_range`	[-90:0.25:89.75]
`hough_thresh`	0.4
`hough_search_r`	7
`AR`	4/3
`fovy`	42.65 deg.
`znear`	0.1
`zfar`	10
`im_size`	[768 1024]

The test images were generated by the following process:

1. Generate the desired trajectory in MATLAB using the vehicle dynamics, a set of control inputs, and a set of initial conditions.
2. Output one element at a time from the resulting sequence of state vectors to FlightGear over a UDP interface using a custom-developed C++ program.
3. For each frame, capture the render window and save it to a PNG file.

Eq. 4.1 shows the formula used for computing the runway's world coordinates in terms of the runway length, *l*, width, *w*, true (not magnetic) heading angle, *Ψ*, and altitude, *h*.

$$\vec{p}_{3D} = \left\{ \begin{array}{c} \langle 0, 0, h \rangle, \\ \left\langle w \cdot \cos\left(\Psi + \frac{\pi}{2}\right), w \cdot \sin\left(\Psi + \frac{\pi}{2}\right), h \right\rangle, \\ \left\langle l \cdot \cos\Psi + w \cdot \cos\left(\Psi + \frac{\pi}{2}\right), l \cdot \sin\Psi + w \cdot \sin\left(\Psi + \frac{\pi}{2}\right), h \right\rangle, \\ \langle l \cdot \cos\Psi, l \cdot \sin\Psi, h \rangle \end{array} \right\} \qquad [4.1]$$

The specific values used for the runway geometry are given in Table 4.2.

Table 4.2: KSFO 19R Runway Properties

Parameter	Value
l (ft.)	7500
w (ft.)	200
Ψ (deg.)	207
h (ft. MSL)	0

4.1.2 Synthetic Test 1: Straight Approach

The first test consists of a straight-in approach with a 5 degree descent at 100 ft./sec. The initial conditions for this test are shown in Table 4.3 for both the true trajectory and the initial estimate for the solver. The rest of the trajectory was simulated at a sampling time of 0.5 sec. until 9.5 secs.

Table 4.3: Initial Conditions for Synthetic Test 1

	True Value	Initial Guess
Φ (deg.)	0	0
Θ (deg.)	-5	-10
Ψ (deg.)	207	210
b_x (ft.)	1203.7	0
b_y (ft.)	501.1	0
b_z (ft.)	-200	-100
U (ft./sec.)	100	80

Figure 4.1 shows the set of images that was generated in FlightGear.

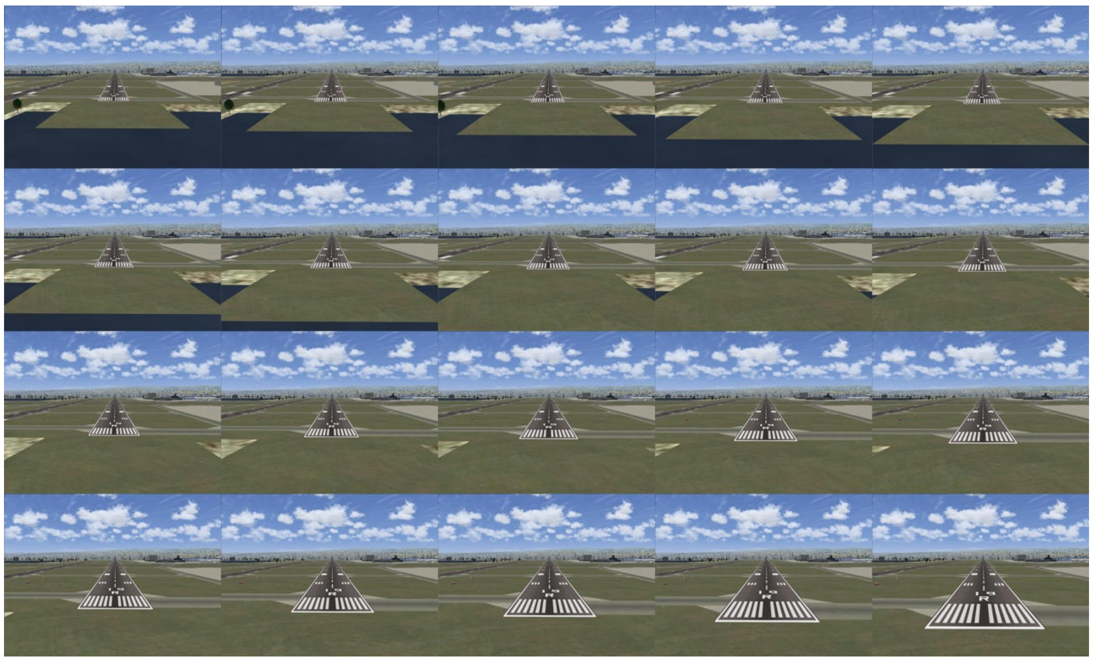


Figure 4.1: Straight Approach Image Set (Synthetic Test 1)

Figure 4.2 shows the results from the Euler angle estimates at each time interval.

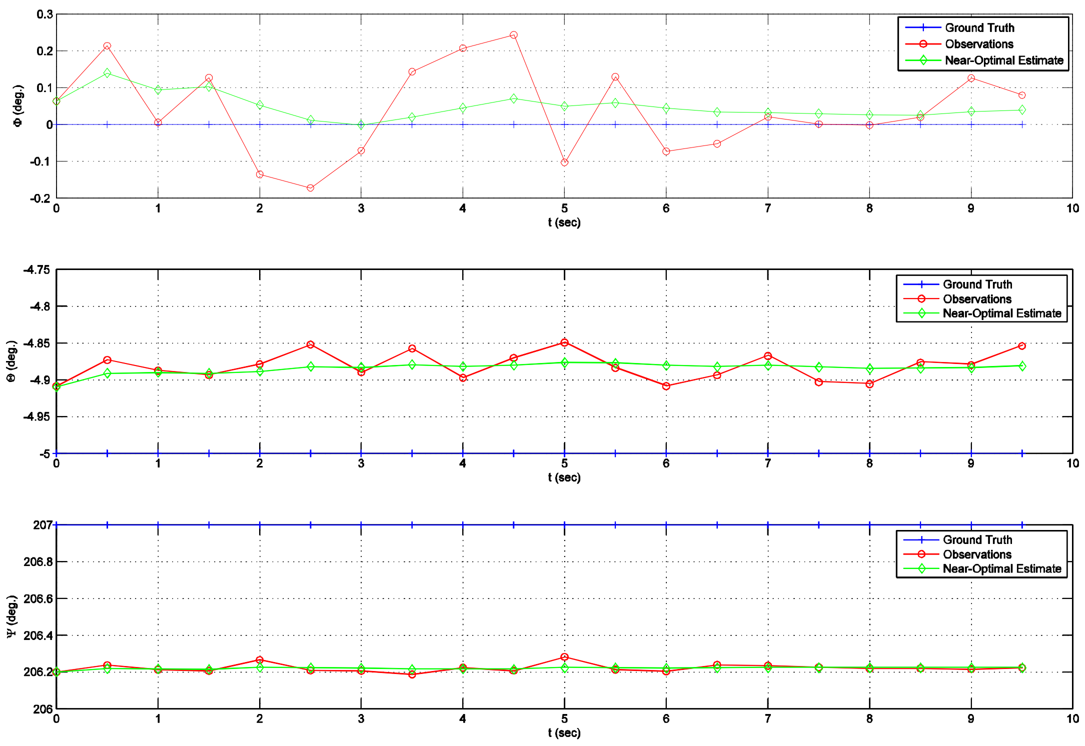


Figure 4.2: Euler Angles from Synthetic Test 1

The trajectory from the simulation is shown in Figure 4.3. Note that the observations and near-optimal estimates are very nearly co-linear, and as such are difficult to distinguish.

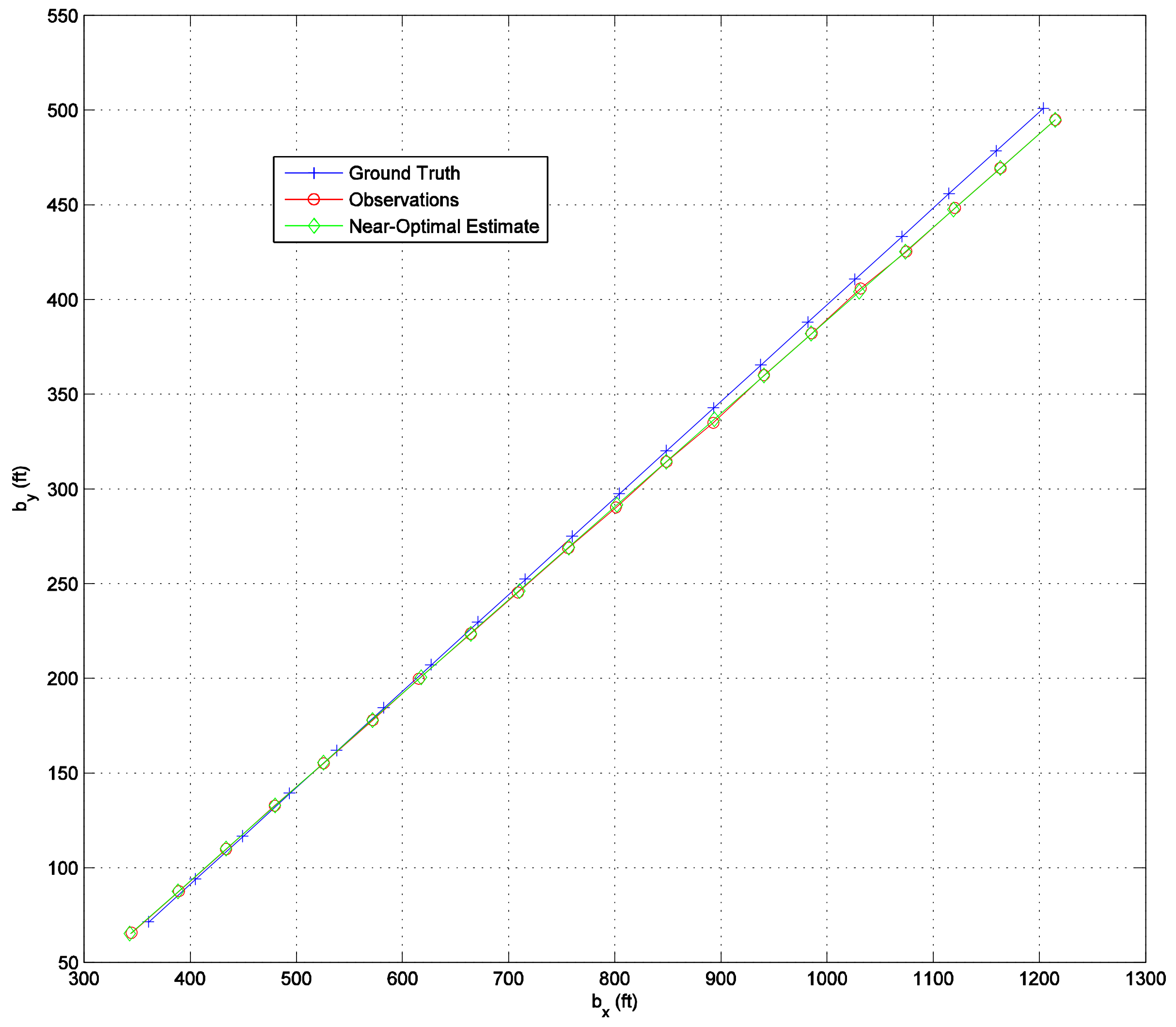


Figure 4.3: X-Y Plane Trajectory from Synthetic Test 1

The estimates for b_z and U are shown in Figure 4.3.

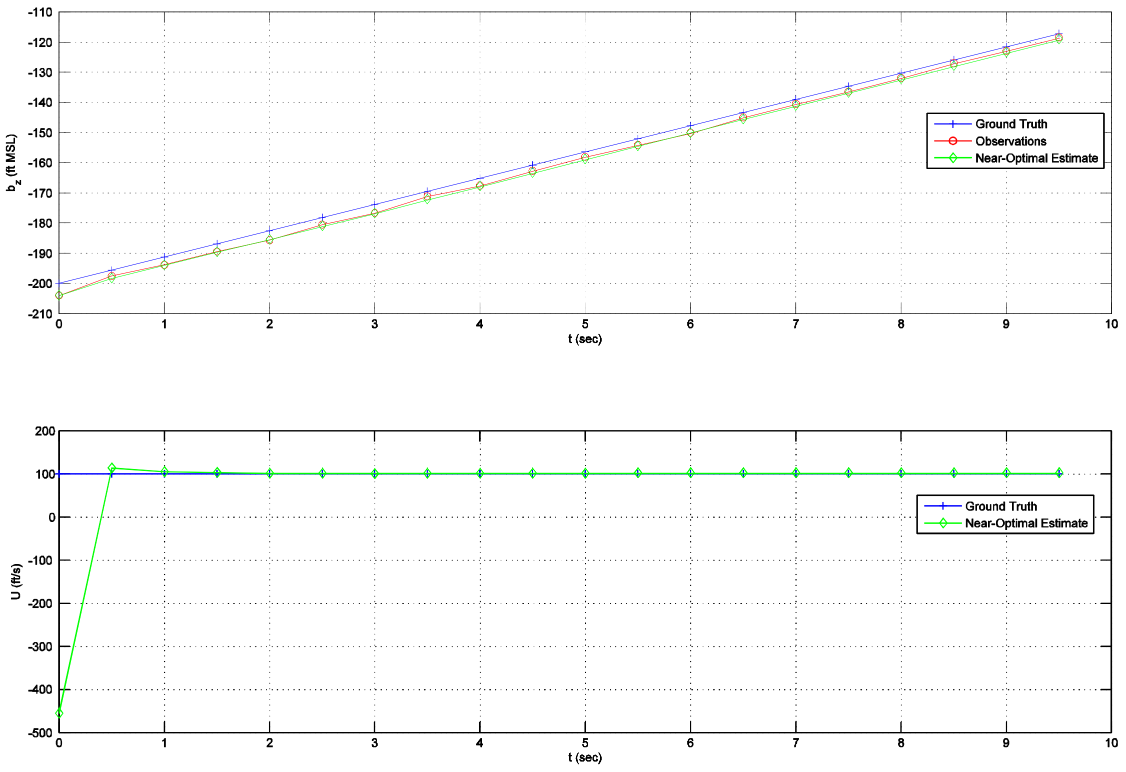


Figure 4.4: Z-Coordinate and Total Velocity from Synthetic Test 1

The trace of the estimate covariance matrix is shown in Figure 4.5. This demonstrates that the Kalman Filter is converging as measurements are accumulated.

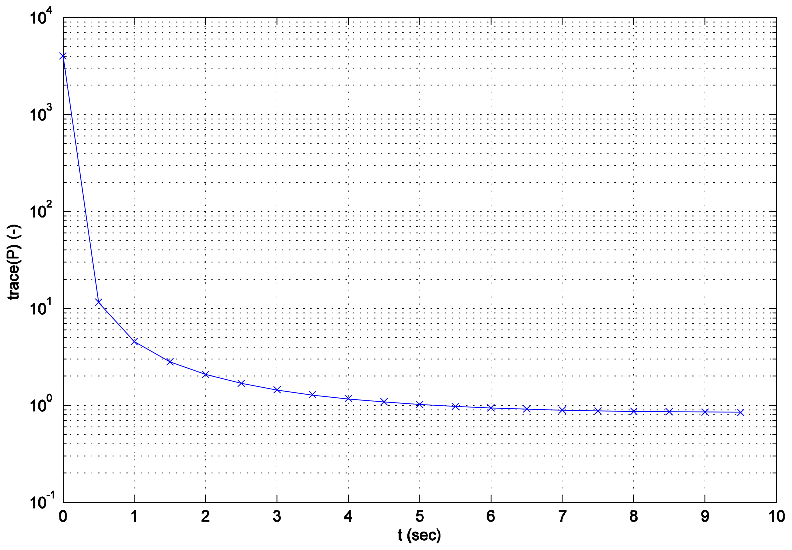


Figure 4.5: EKF Covariance Matrix Trace

Figure 4.6 shows in better detail the error in each of the Euler angle estimates. These errors are defined as the difference between the observations or near-optimal estimates and the ground truth – i.e., 0 corresponds to no error. The estimates for roll angle are never worse than 0.15 degrees and the error in pitch is never worse than 0.13 degrees. The error in heading, however, has a fairly constant bias of almost 0.8 degrees. This is likely due to an incorrect assumption about how the runway is rendered in FlightGear.

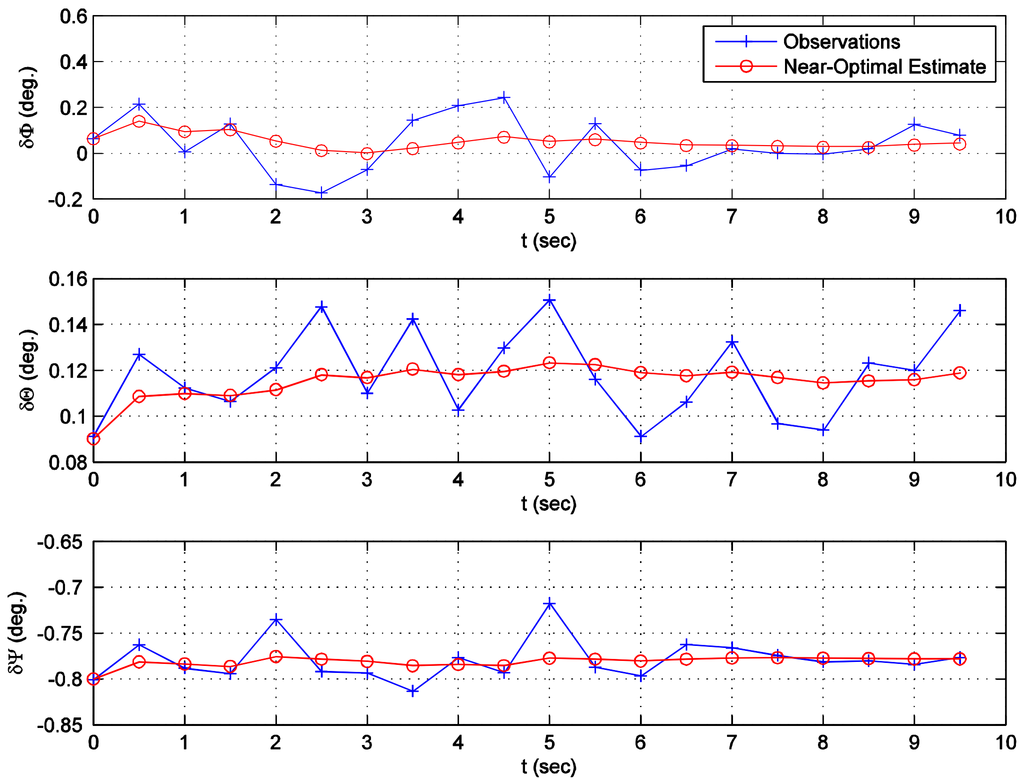


Figure 4.6: Euler Angle Accuracy in Synthetic Test 1

The position accuracy is shown in Figure 4.7. The results indicate that there is in fact an incorrect assumption regarding either the field-of-view angle or the rendering of the runway, most probably the latter. Despite these defects, the accuracy of the algorithm still beats the accuracy of GPS and succeeds in measuring altitude to within 2 feet toward the end of the simulation.

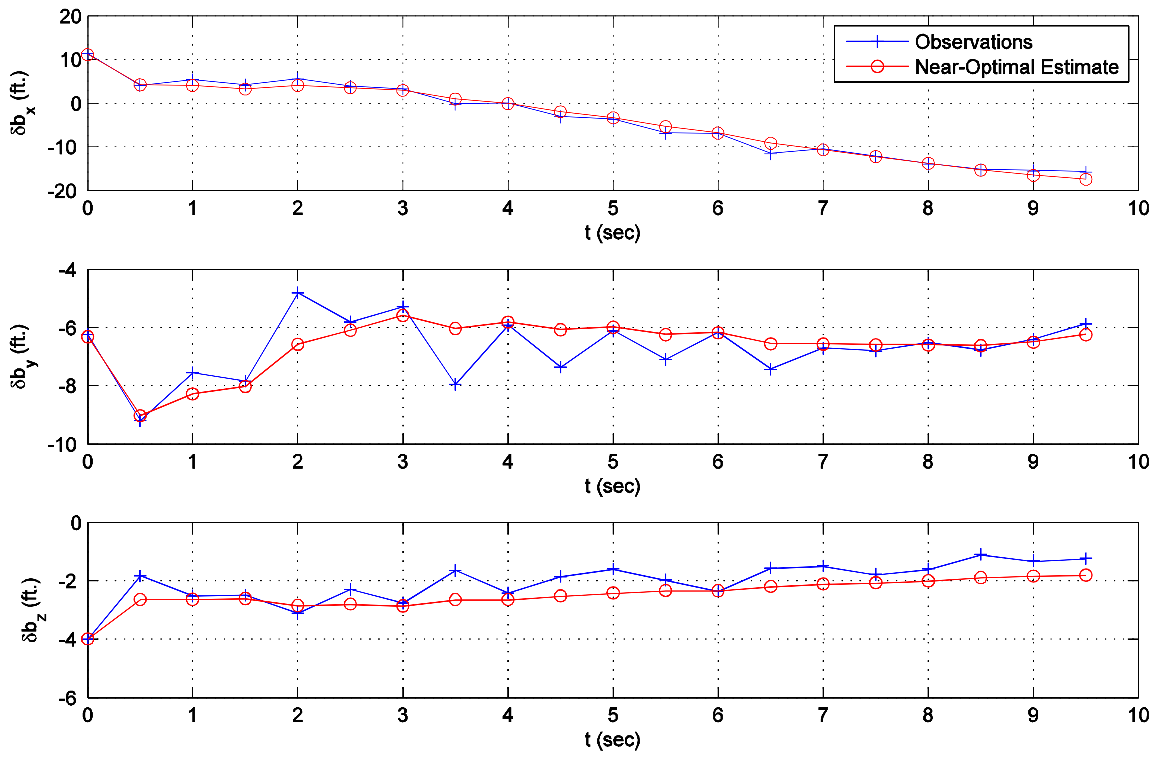


Figure 4.7: Position Accuracy in Synthetic Test 1

4.1.3 Synthetic Test 2: Pull-Up Maneuver on Approach

The second test consists of a gentle pull-up maneuver during an approach. Table 4.4 shows the initial conditions used for the test.

Table 4.4: Initial Conditions for Synthetic Test 2

	True Value	**Initial Guess**
Φ (deg.)	0	0
Θ (deg.)	-20	-10
Ψ (deg.)	207	210
b_x (ft.)	1203.7	0
b_y (ft.)	501.1	0
b_z (ft.)	-200	-100
U (ft./sec.)	100	80

The control inputs used to simulate the pull-up maneuver are shown in Eq. 4.2.

$$u_{\Theta} = \begin{cases} 3^\circ/\sec & \textit{if } \; t \le 6 \; \sec \\ 0 & \textit{otherwise} \end{cases} \qquad [4.2]$$

Figure 4.8 shows the test image set that was generated using FlightGear.

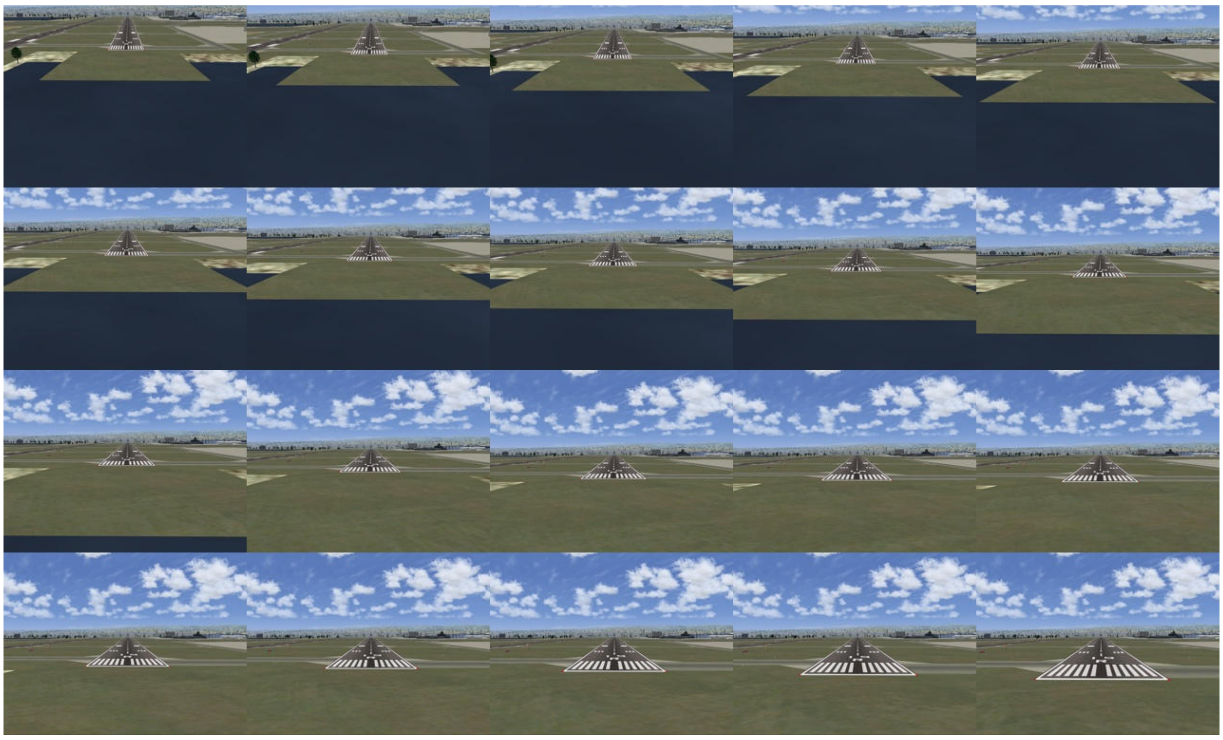


Figure 4.8 Pull-Up Maneuver Image Set (Synthetic Test 2)

Figure 4.9 shows the Euler angle estimates from this image set.

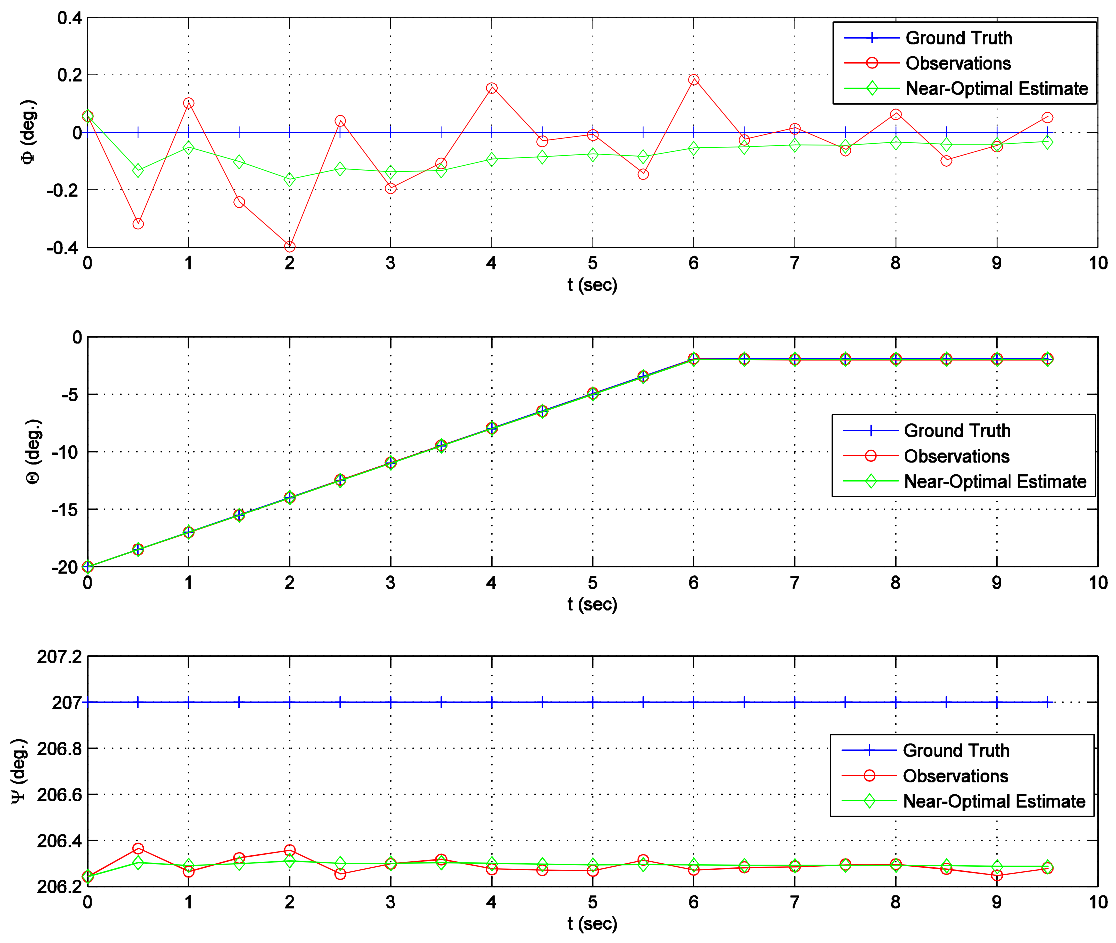


Figure 4.9: Euler Angles from Synthetic Test 2

Figure 4.10 shows the estimated X-Y plane trajectory from this test.

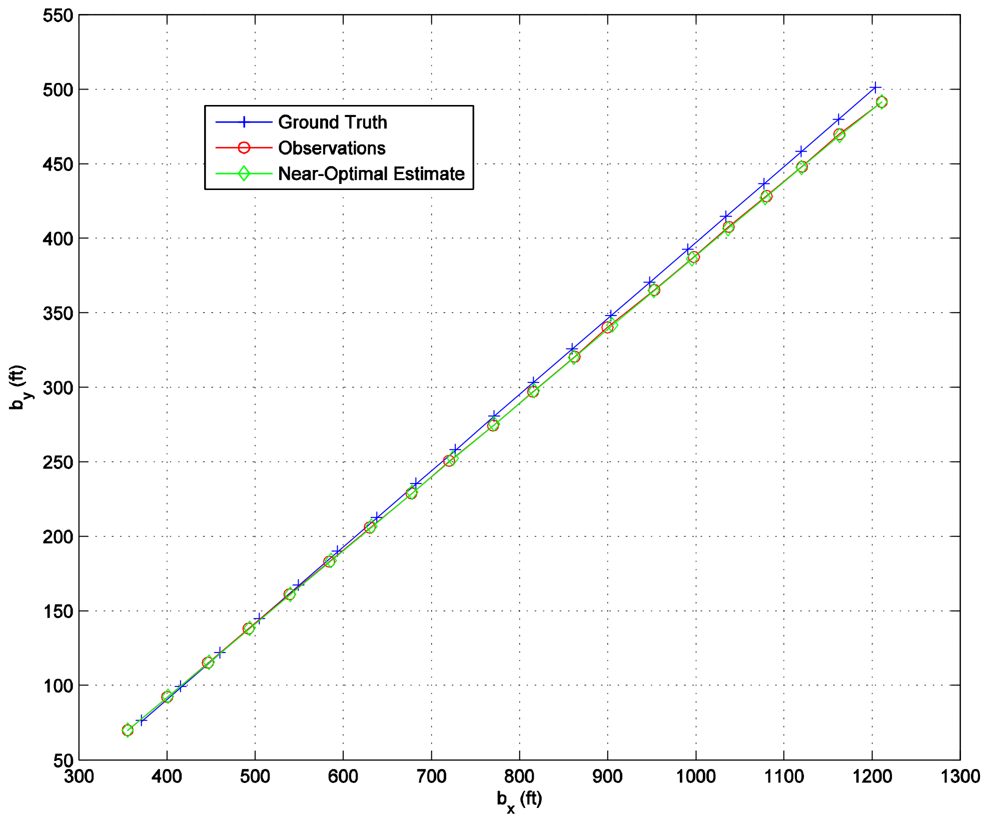


Figure 4.10: X-Y Plane Trajectory from Synthetic Test 2

Figure 4.11 shows the b_z and U estimates from this test.

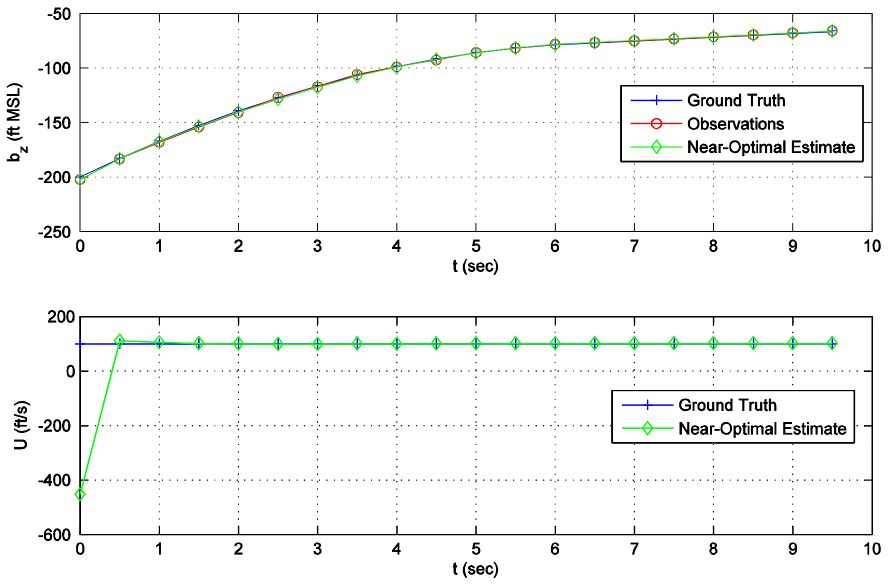


Figure 4.11: Z-Coordinate and Total Velocity from Synthetic Test 2

Figure 4.12 shows the state estimate covariance matrix trace. Again, the filter converges.

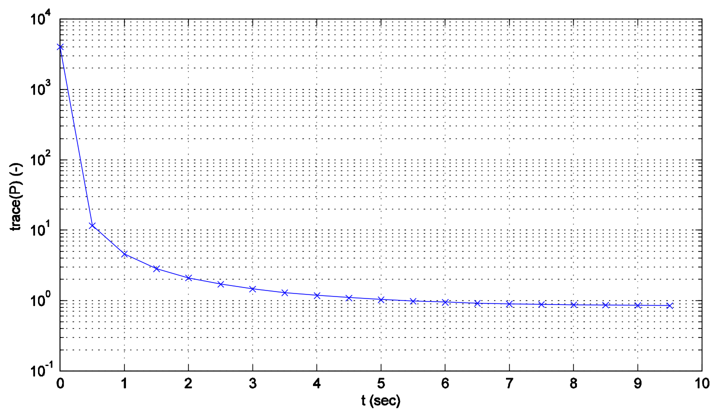


Figure 4.12: EKF Covariance Matrix Trace from Synthetic Test 2

Figure 4.13 shows the Euler angle accuracy from this test. Once again very good accuracy is observed, even with a changing pitch angle. However the heading angle bias is again observed, further pointing to an error in the assumption of how the runway is rendered.

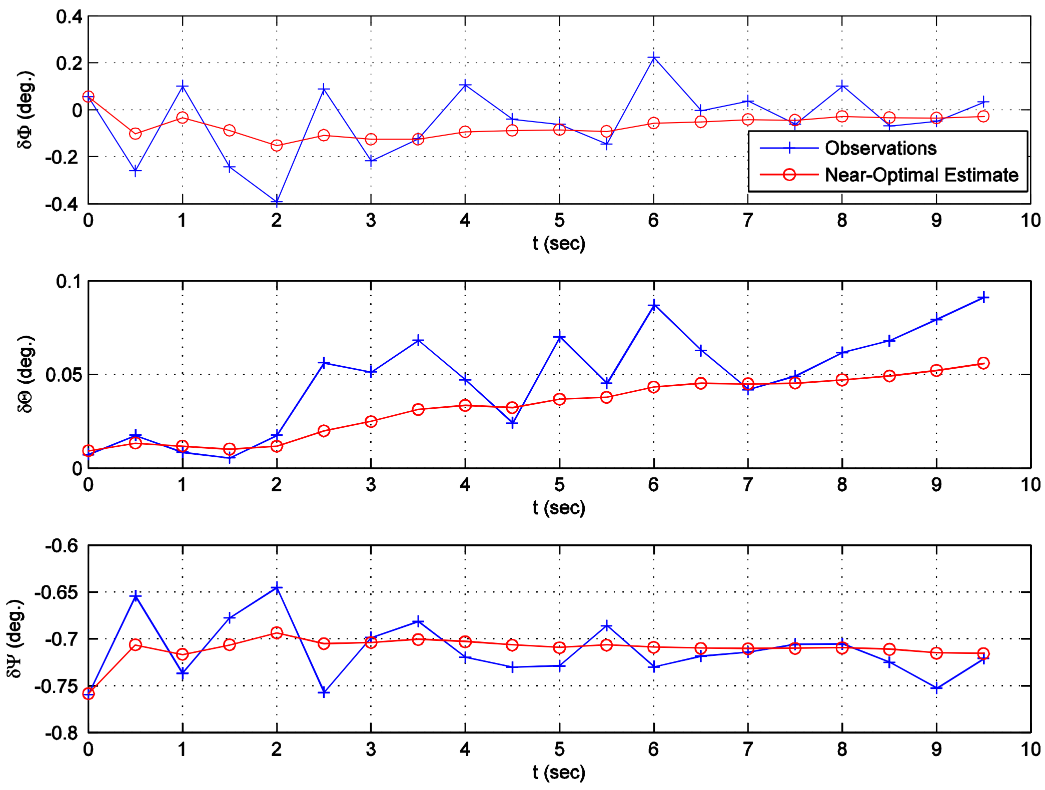


Figure 4.13: Euler Angle Accuracy in Synthetic Test 2

Figure 4.14 shows the accuracy in the position estimate in this test. Similar trends are observed in this test as in the previous test, but altitude accuracy is nearly “dead-on” by the end of the simulation.

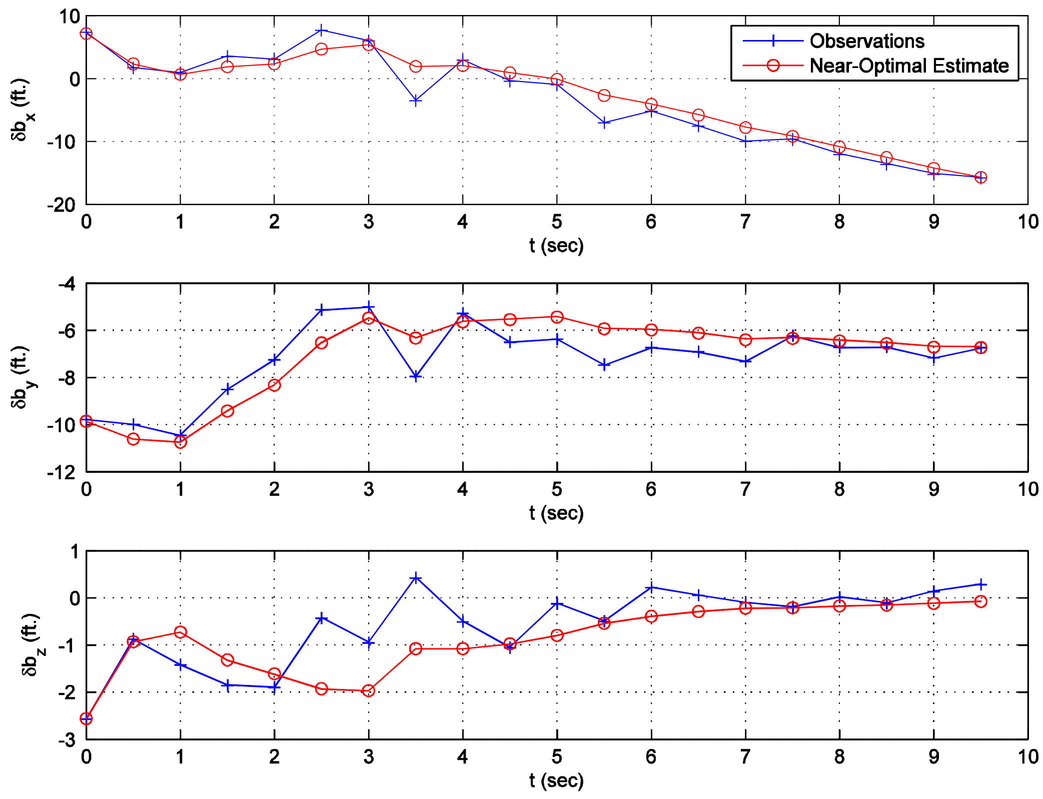


Figure 4.14: Position Accuracy in Synthetic Test 2

4.1.4 Synthetic Test 3: Turn from Base to Final

The final synthetic test consists of a simulation of the latter part of a tight turn from a base leg to final. Table 4.5 shows the initial conditions for this test.

Table 4.5: Initial Conditions for Synthetic Test 3

	True Value	**Initial Guess**
Φ (deg.)	-30	0
Θ (deg.)	-5	-10
Ψ (deg.)	227	210
b_x (ft.)	1158.3	0
b_y (ft.)	590.2	0
b_z (ft.)	-200	-100
U (ft./sec.)	100	80

Eqs. 4.3 and 4.4 show the control inputs used to generate the desired trajectory.

$$u_\Phi = \begin{cases} 12^\circ/\sec & if\ \ 4\ \sec \le t \le 6.5\ \sec \\ 0 & otherwise \end{cases} \qquad [4.3]$$

$$u_\Psi = \begin{cases} -4^\circ/\sec & if\ \ t \le 5\ \sec \\ 0 & otherwise \end{cases} \qquad [4.4]$$

Figure 4.15 shows the image set generated.

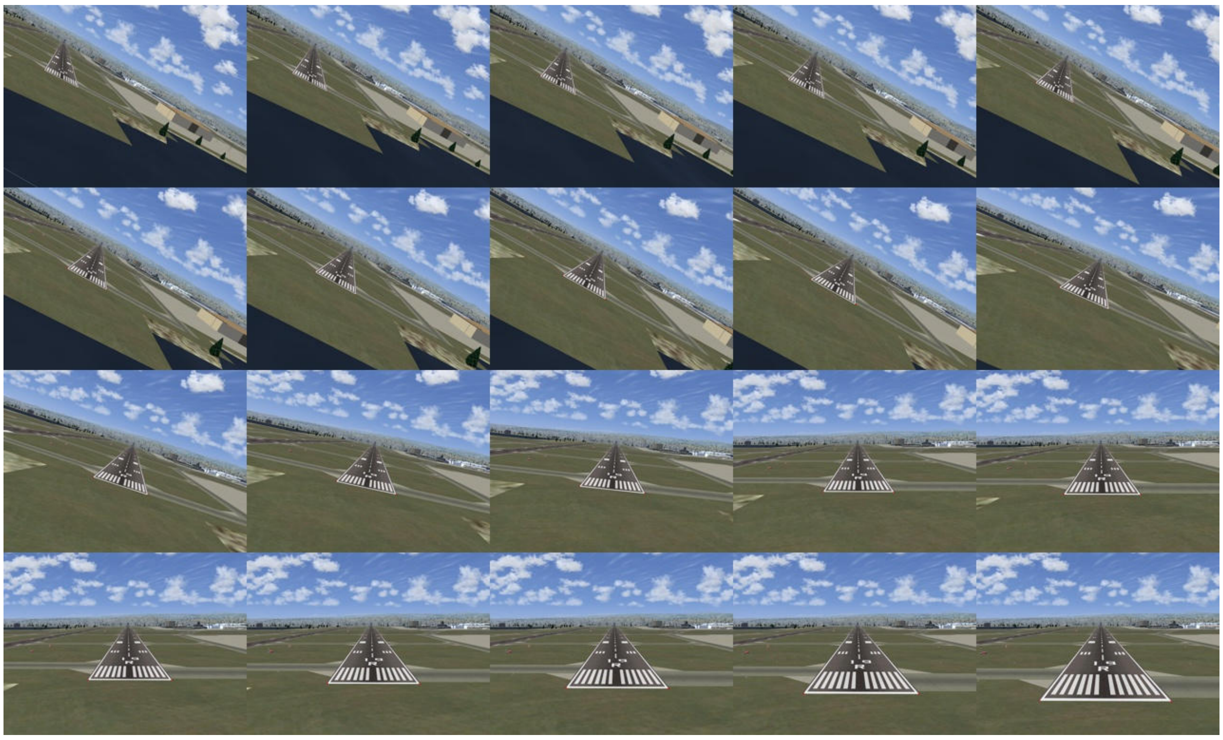


Figure 4.15: Base to Final Image Set (Synthetic Test 3)

Figure 4.16 shows the estimates for the Euler angles for this test. The tracking was very good in this test for all angles except heading, which exhibited a small bias. The EKF performed very well in tracking roll angle, showing very small estimation errors are no obvious phase delay in tracking the time-varying part of the roll angle history from *t = 4 sec* to *t = 6.5 sec*.

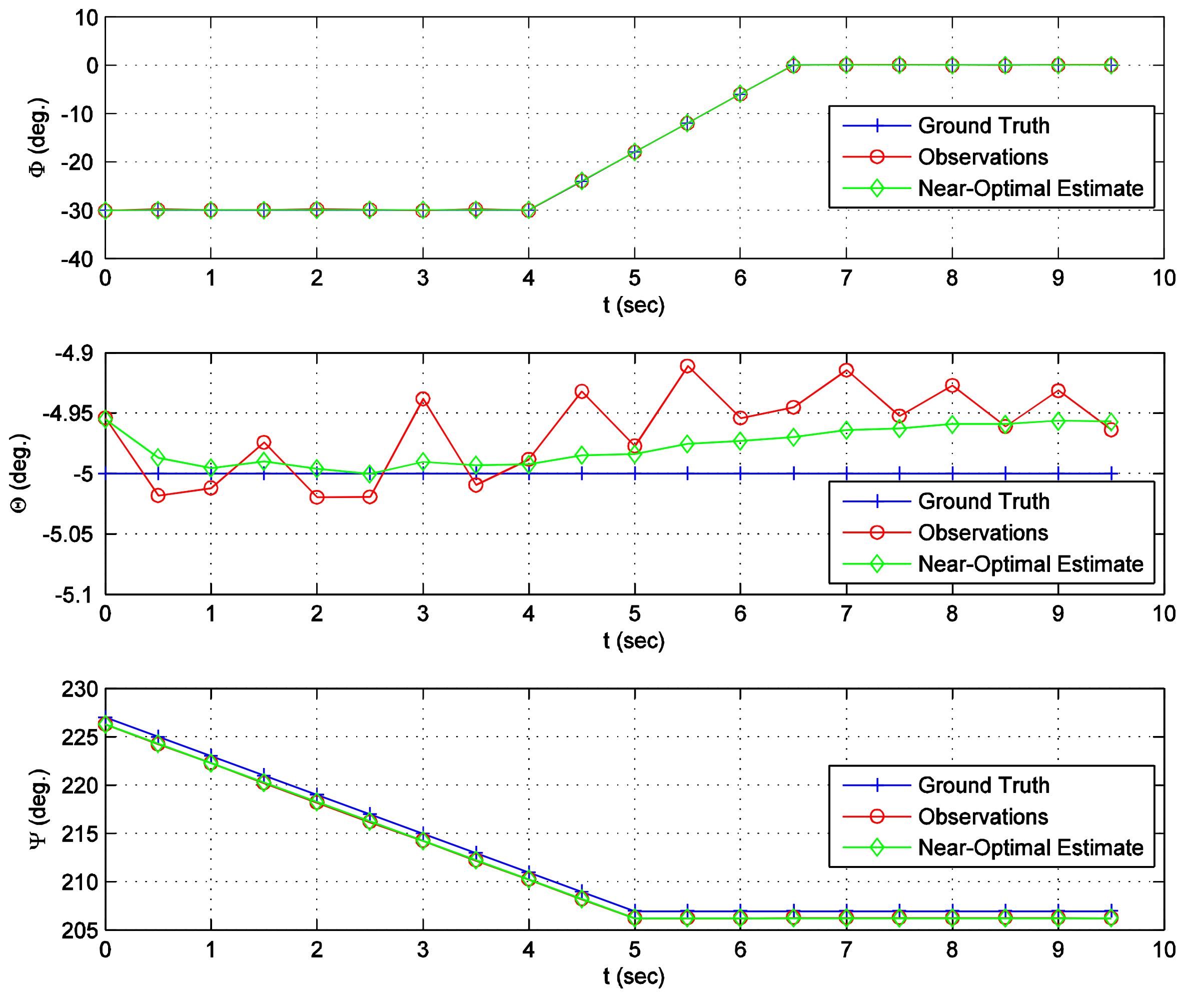


Figure 4.16: Euler Angles from Synthetic Test 3

Figure 4.17 shows the X-Y plane trajectory from the third synthetic test. Tracking was good, but there is clearly some substantial error between the ground truth and the estimates.

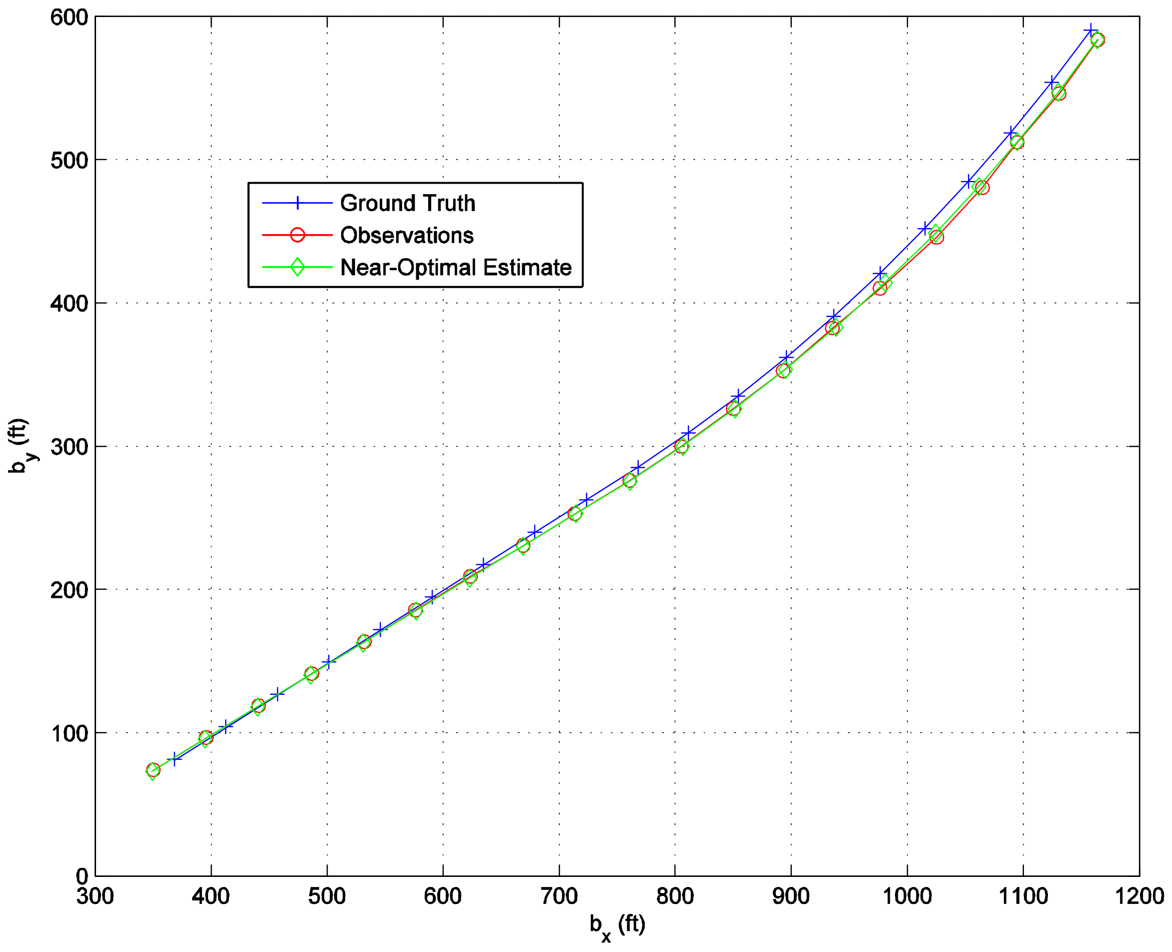


Figure 4.17: X-Y Plane Trajectory from Synthetic Test 3

Figure 4.18 shows the estimates for b_z and U.

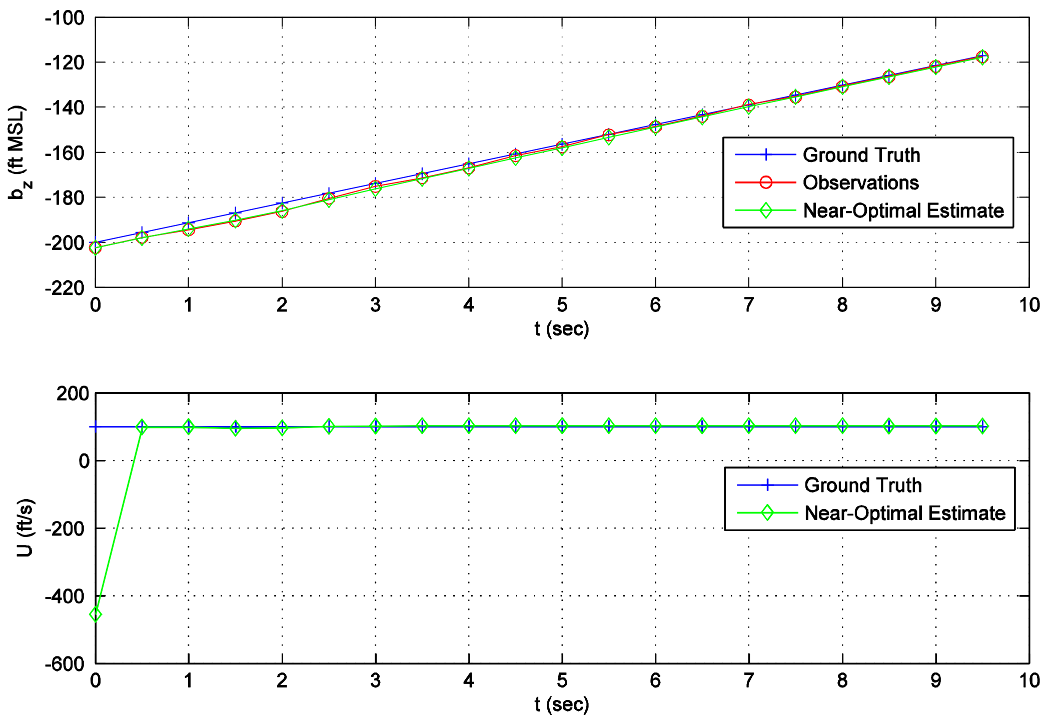


Figure 4.18: Z-Coordinate and Total Velocity for Synthetic Test 3

Figure 4.19 shows the convergence of the Kalman filter.

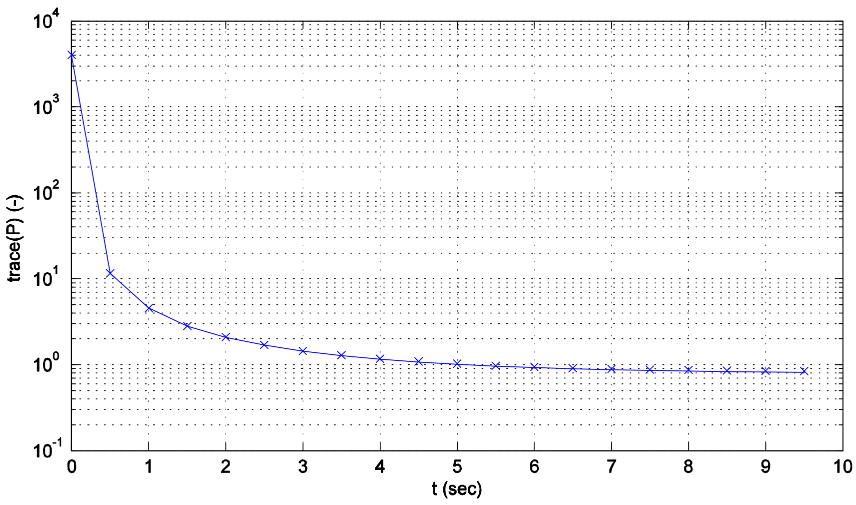


Figure 4.19: EKF Covariance Matrix Trace for Synthetic Test 3

The accuracy for the Euler angle estimations is shown in Figure 4.20. Once again, very good accuracy is observed except for the heading angle estimation, which is systematically off by -0.7 to -0.8 degrees.

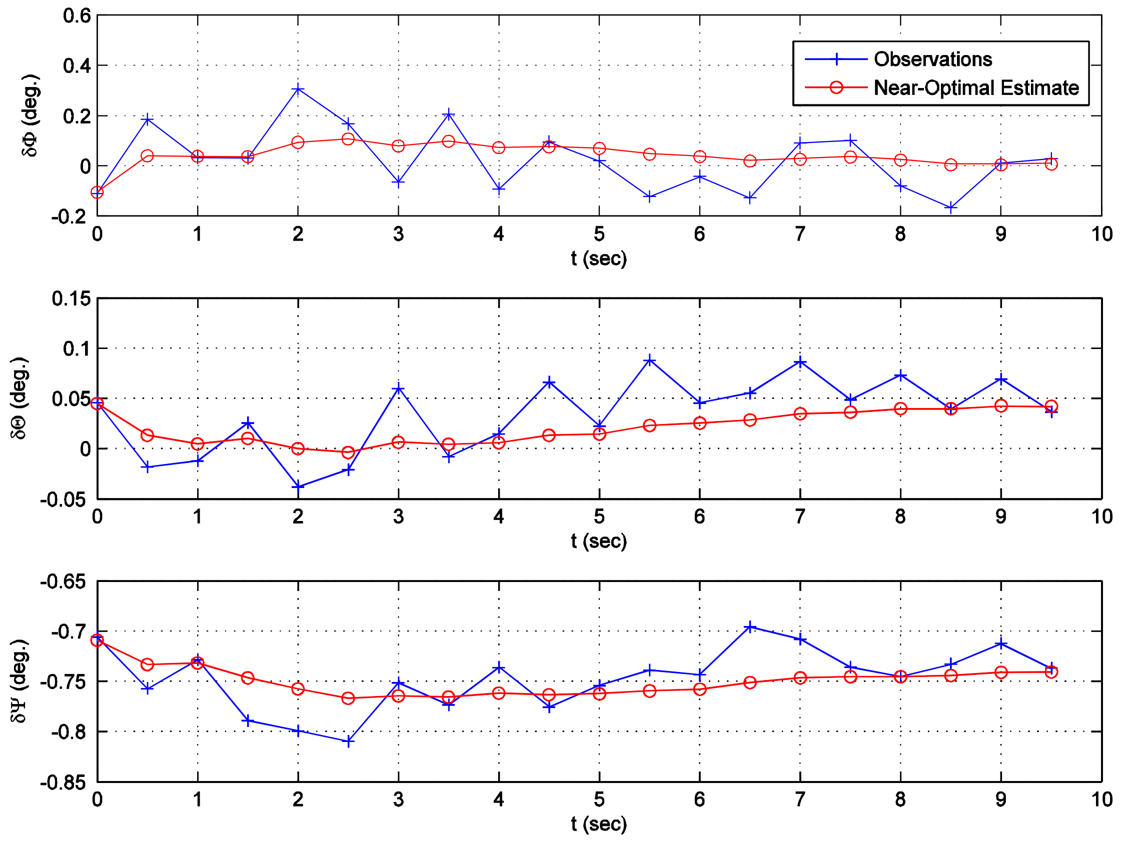


Figure 4.20: Euler Angle Accuracy in Synthetic Test 3

Similar trends in position accuracy are observed in Figure 4.21. Altitude estimation – the most critical parameter for a UAV autolanding system – remains very good.

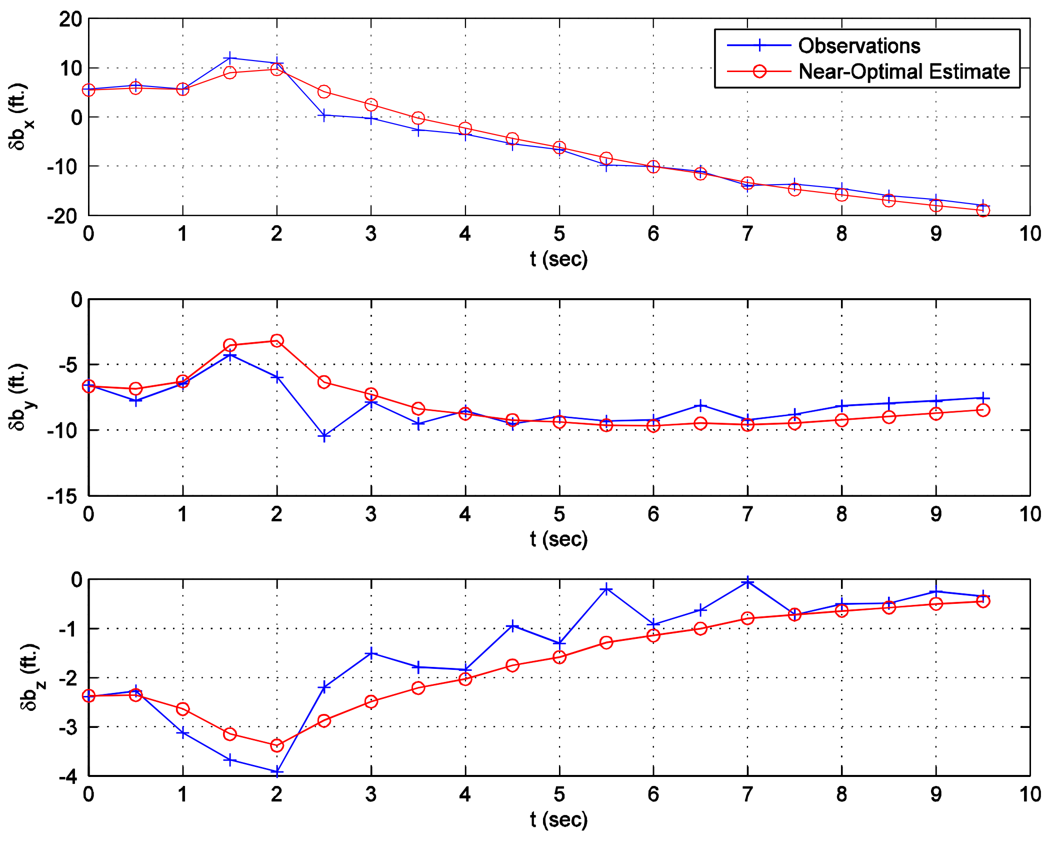


Figure 4.21: Position Accuracy in Synthetic Test 3

4.2 Natural Image Test

4.2.1 Test Setup

This test was performed on a model runway by sliding a camera along a fixed rail and taking images at regular intervals to simulate motion. Due to the difficulty of using such a rail system without obstructing the view of the camera, a 15.1 degree wedge was used to elevate the camera out of the view of the rail and a compensating "angle-of-attack" term was added to the system dynamics to allow for an angle between the glidepath angle and the true pitch angle.

The rail system allows for a constant roll angle, pitch angle, heading angle, and lateral runway offset, and a linearly changing trajectory for altitude and distance to runway. Table 4.6

shows the initial conditions for the natural image test. The full trajectory was generated by taking a snapshot every 2 inches along the rail.

Table 4.6: Initial Conditions for Natural Image Test

	True Value	Initial Guess
Φ (deg.)	4.4	0
Θ (deg.)	-6.2	-10
Ψ (deg.)	0	0
b_x (in.)	-72	0
b_y (in.)	10	-100
b_z (in.)	-25.5	-10
U (in./sec.)	4	80

Table 4.7 shows the configuration properties for the natural image test. These are similar to the properties selected for the synthetic tests, but are modified due to slight differences in the marker and line colors.

Table 4.7: Configuration Parameters in Natural Image Test

Name	Value
`line_thresh`	50^2
`mrkr_thresh`	50^2
`line_color`	[220 230 240]
`mrkr_color`	[200 50 100]
`line_sigma`	1
`mrkr_search_r`	20
`hough_rho_res`	2
`hough_phi_range`	[-90:0.25:89.75]
`hough_thresh`	0.3
`hough_search_r`	7
`AR`	4/3
`fovy`	39.37 deg.
`znear`	0.1
`zfar`	10
`im_size`	[768 1024]

4.2.2 Test Results

Figure 4.22 shows the image set used for this test. As reflected from the tables in the previous section, there is a slight constant roll and pitch angle. The most significant difference is that the markers used for this test are much larger than those used in the synthetic tests. The markers were required to be larger in order the markers located on the far side of the runway be properly detected by the algorithm.

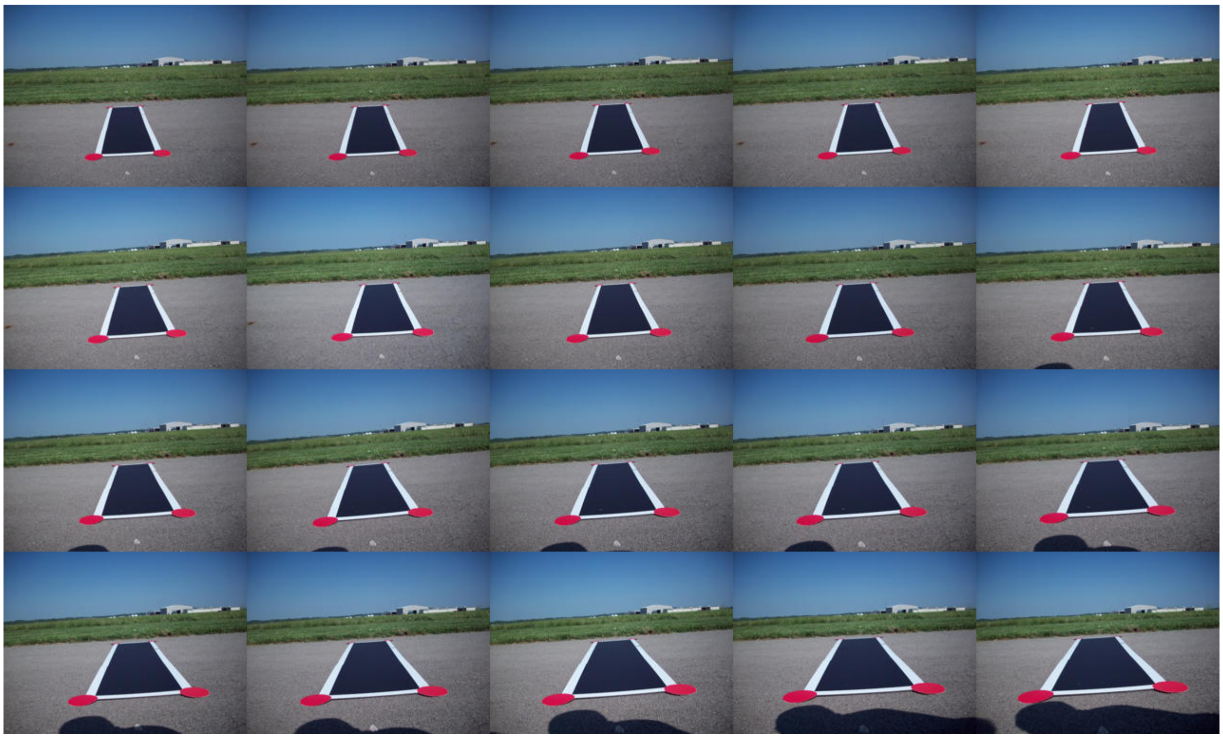


Figure 4.22: Natural Image Data Set

Figure 4.23 shows the Euler angle estimations in the natural image test. Aside from a few poor estimates early in the trajectory, the algorithm performs nearly equally well in Euler angle estimation using natural images as it does in the synthetic case.

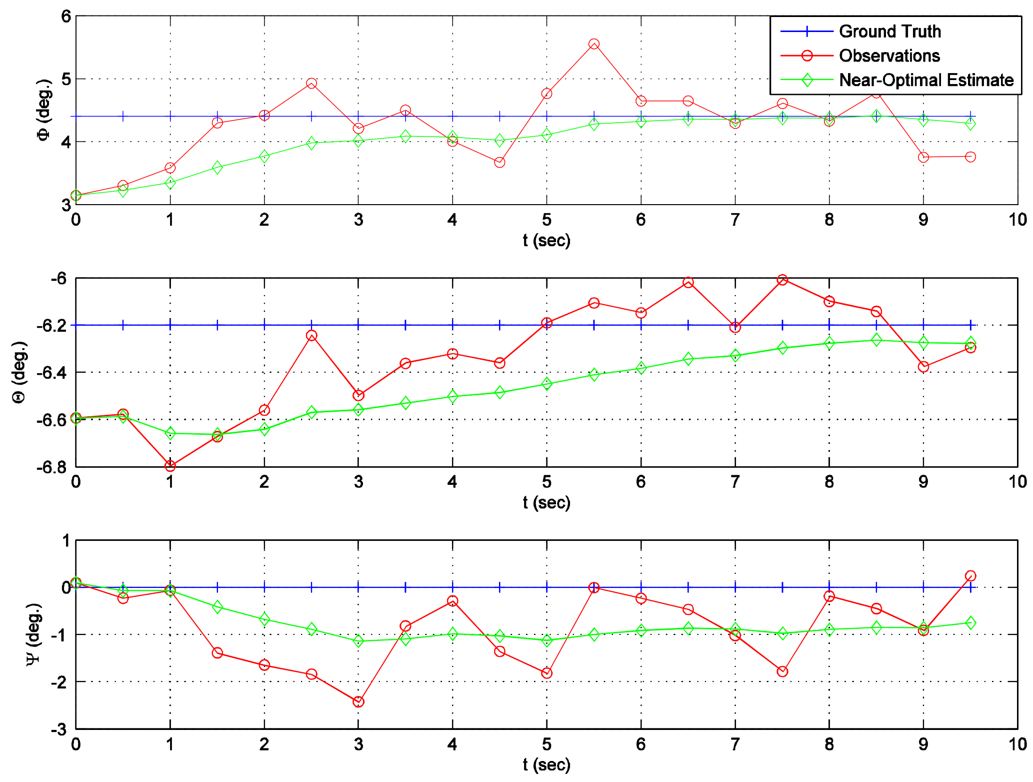


Figure 4.23: Euler Angles from Natural Image Test

The ground plane trajectory is shown in Figure 4.24.

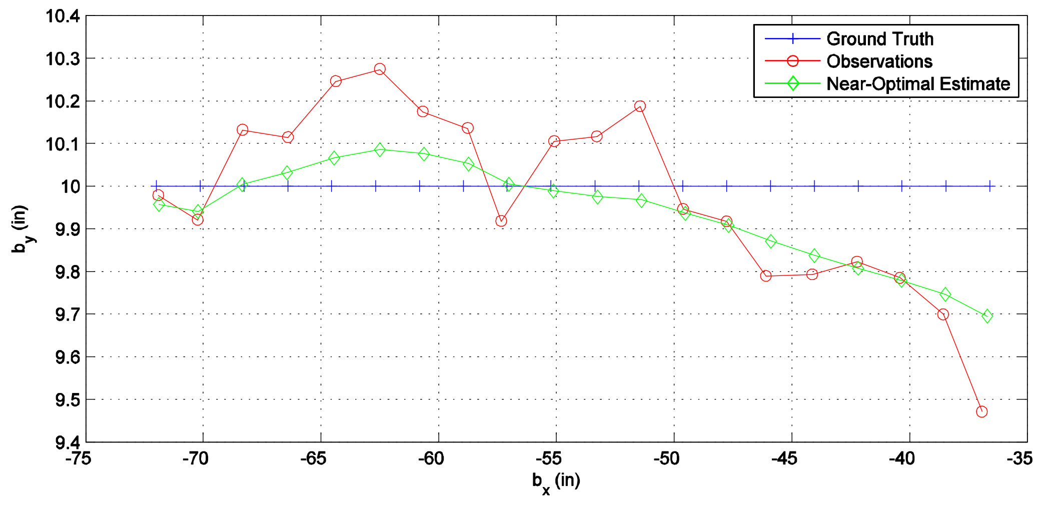


Figure 4.24: Trajectory from Natural Image Test

The estimates for b_z and U are shown in Figure 4.25.

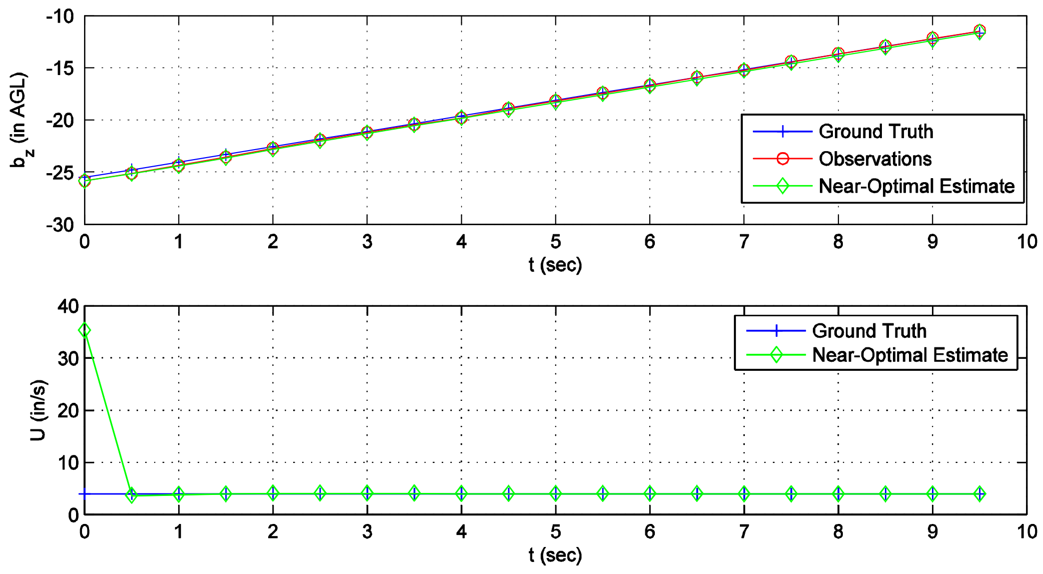


Figure 4.25: Altitude and Total Velocity from Natural Image Test

Figure 4.26 shows that once again the EKF has properly converged by the end of the simulation.

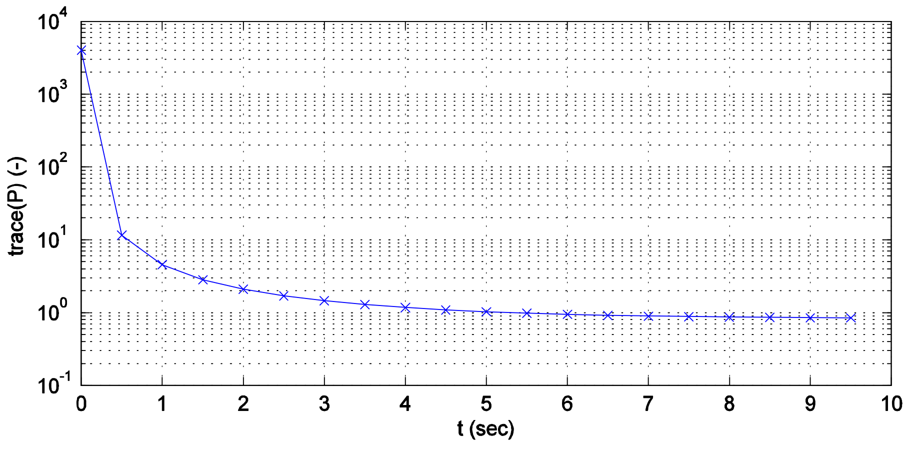


Figure 4.26: Covariance Matrix Trace from Natural Image Test

Figure 4.27 shows the accuracy of the Euler angle estimations. The early inaccuracy in the first few observations can be attributed to initial inexperience of the data gatherers in taking images without affecting the camera's position and orientation. The test apparatus used is of a very low-grade, and it would have been easy to accidentally take images that were not oriented correctly. However, this problem clearly diminishes in later observations. That being said, the -0.8 degree bias in the heading angle estimate appears to be present once again. This likely points to a systematic problem with the implementation of the algorithm.

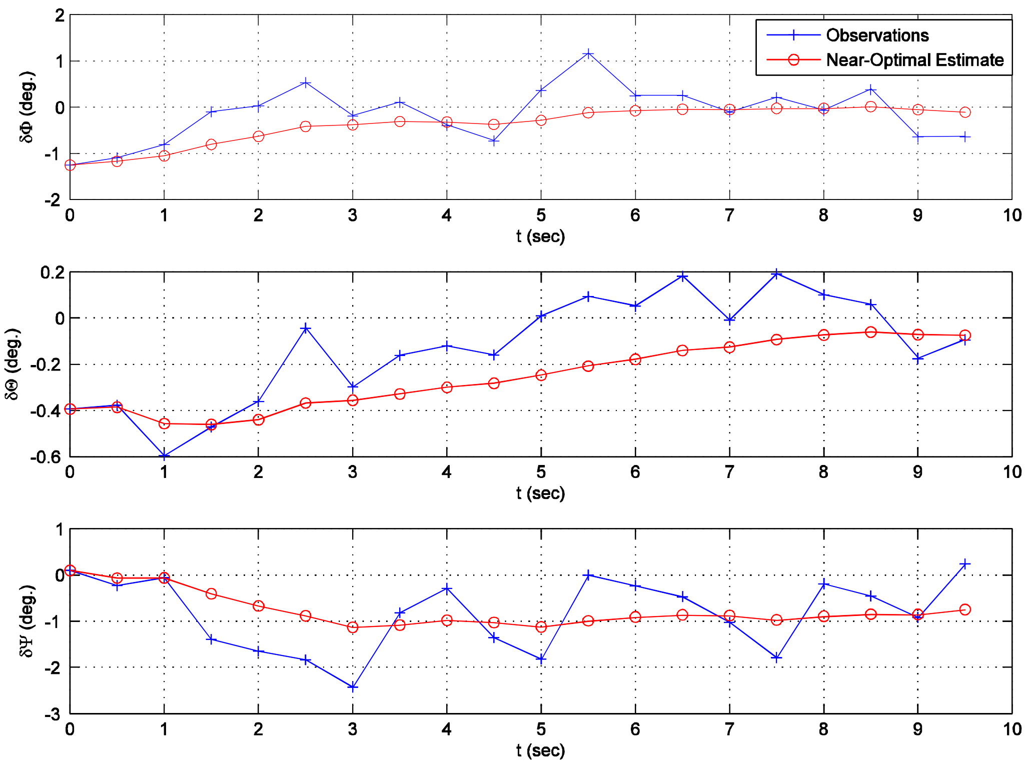


Figure 4.27: Euler Angle Accuracy in Natural Image Test

Figure 4.28 shows the position accuracy in the natural image test. Very good results are obtained across-the-board in each axis with accuracy to within ¼” except for the first few altitude measurements.

It should be noted that the accuracy of the algorithm is probably much higher than the accuracy of the ground truth. Since these measurements were performed by hand using a crude rail system, it is entirely possible that much more accurate results would be seen using a more robust apparatus.

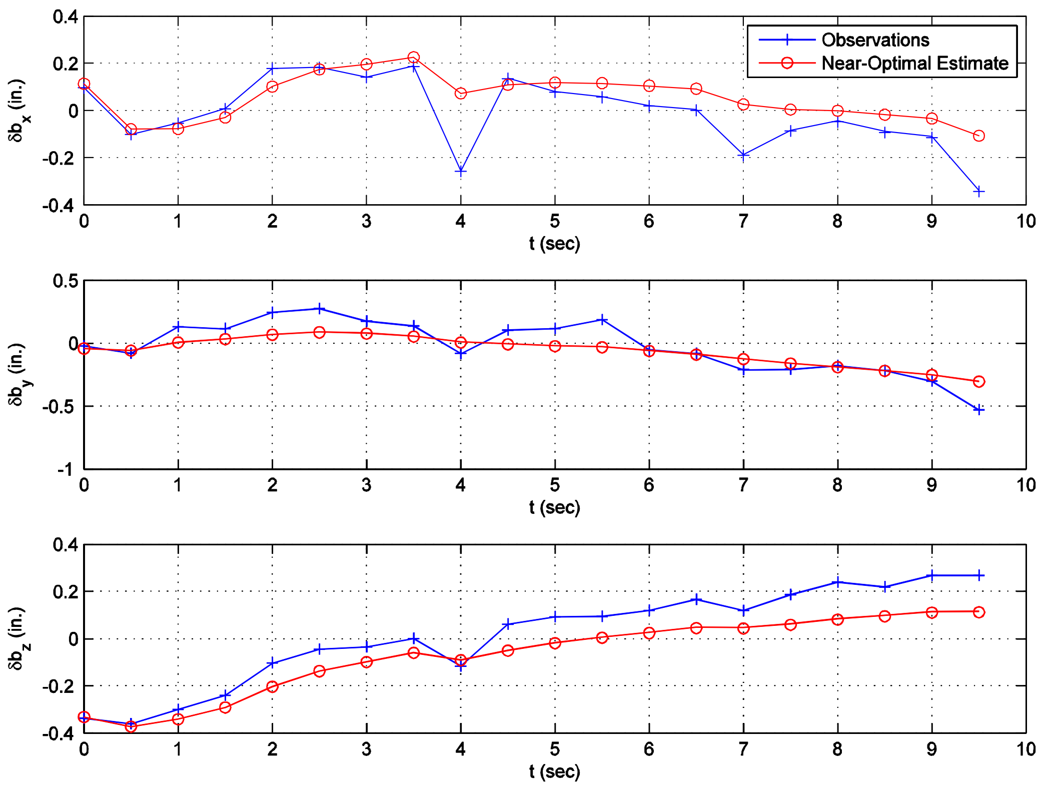


Figure 4.28: Position Accuracy in Natural Image Test

4.3 Sensitivity Analysis

4.3.1 Overview

A sensitivity analysis is performed on a set of synthetic and natural images to determine the effect of the accuracy of the primary inputs to the state observer on the overall state observation. For the case which is of primary interest in this Thesis, the most important measurements are:

- The 4 $<x_{im}, y_{im}>$ pairs which define the outline of the runway, in pixels.
- The field-of-view angle in the *y*-direction, fov_y, in degrees.

Other parameters such as the width of the runway, length of the runway, runway orientation, elevation, etc. are assumed to be known well enough that any estimate errors resulting from a normal range of measurement errors in these parameters are much less than those that would result from comparable errors in the corner detection and field-of-view angle measurement.

The sensitivities are numerically calculated by computing the effect of a small perturbation about the estimated state. This method assumes that the projection equations are locally linear in the range of the perturbation, which is reasonable to assume given the rapidity of the convergence of the solver.

In order to condense the amount of data that must be sifted through to understand the overall sensitivity, the root-mean-square (RMS) sensitivity of the corner measurement is computed rather than the individual sensitivity.

4.3.2 Synthetic Tests

The sensitivity of Euler Angles to corner measurement is shown in Figure 4.29. It is clear from this figure that the algorithm was not especially sensitive to corner measurement for angle estimation.

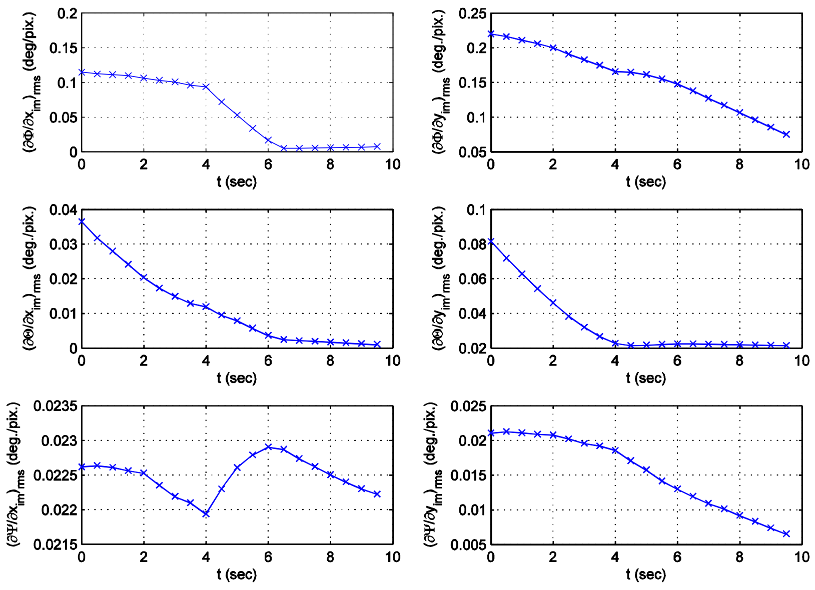


Figure 4.29: Sensitivity of Euler Angle Estimates to RMS Corner Measurement Error in Synthetic Test 3

The sensitivity of position estimates to RMS corner measurement is shown in Figure 4.30. The position errors are much more sensitive to corner measurement than are the angle estimates. A 5 pixel error in measurement would lead to 25 feet of X-Y plane position error at the start of the trajectory. However, this error diminishes as the runway comes into larger view in the image. Altitude sensitivity is much more important than X-Y plane position, and this sensitivity is much lower. It must be recognized that the error in any plane will be a function of the approach angle. It can be observed from this graph that altitude estimation is favorable to a shallow approach angle.

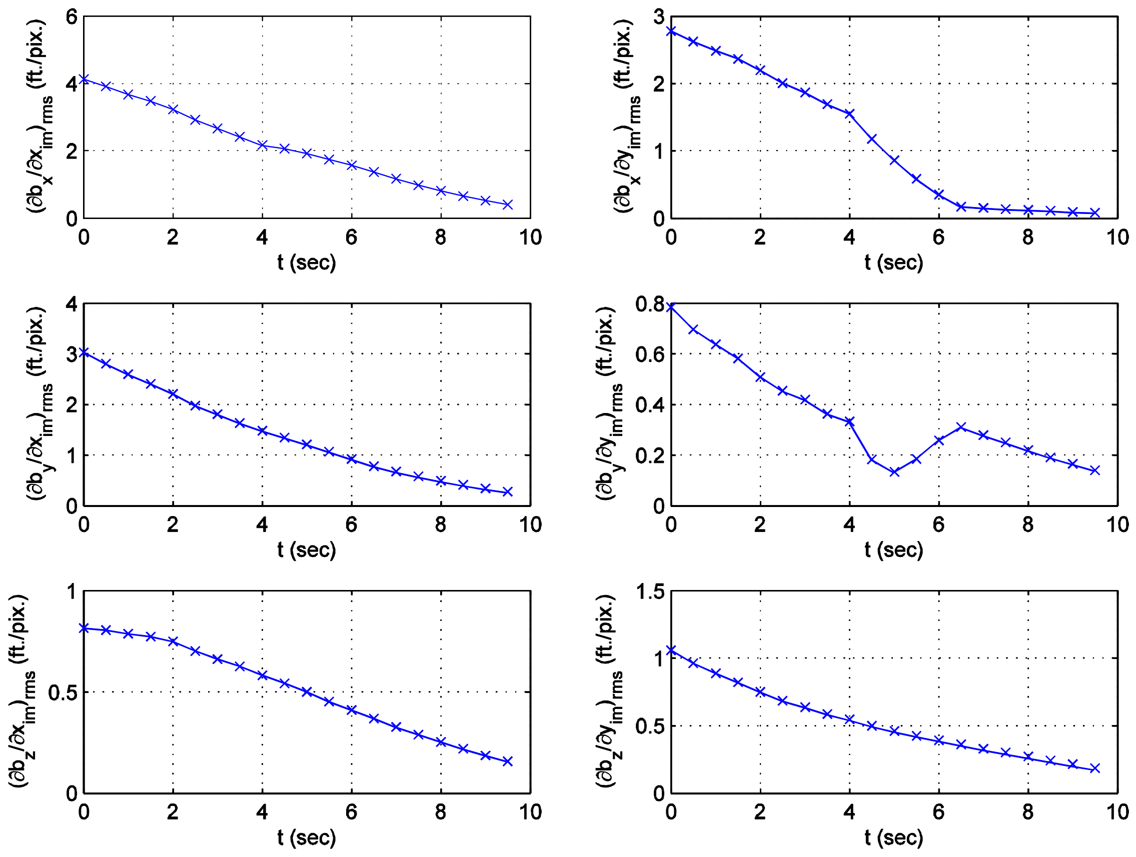


Figure 4.30: Sensitivity of Position Estimates to RMS Corner Measurement Error in Synthetic Test 3

Figure 4.31 shows the sensitivity of the state estimate to field-of-view angle. While the effect of the field-of-view angle on Euler angle estimates is negligible, it has a profound effect on the estimates for position, especially in the X-Y plane. In all cases, the sensitivity to measurement errors diminishes as the runway becomes closer.

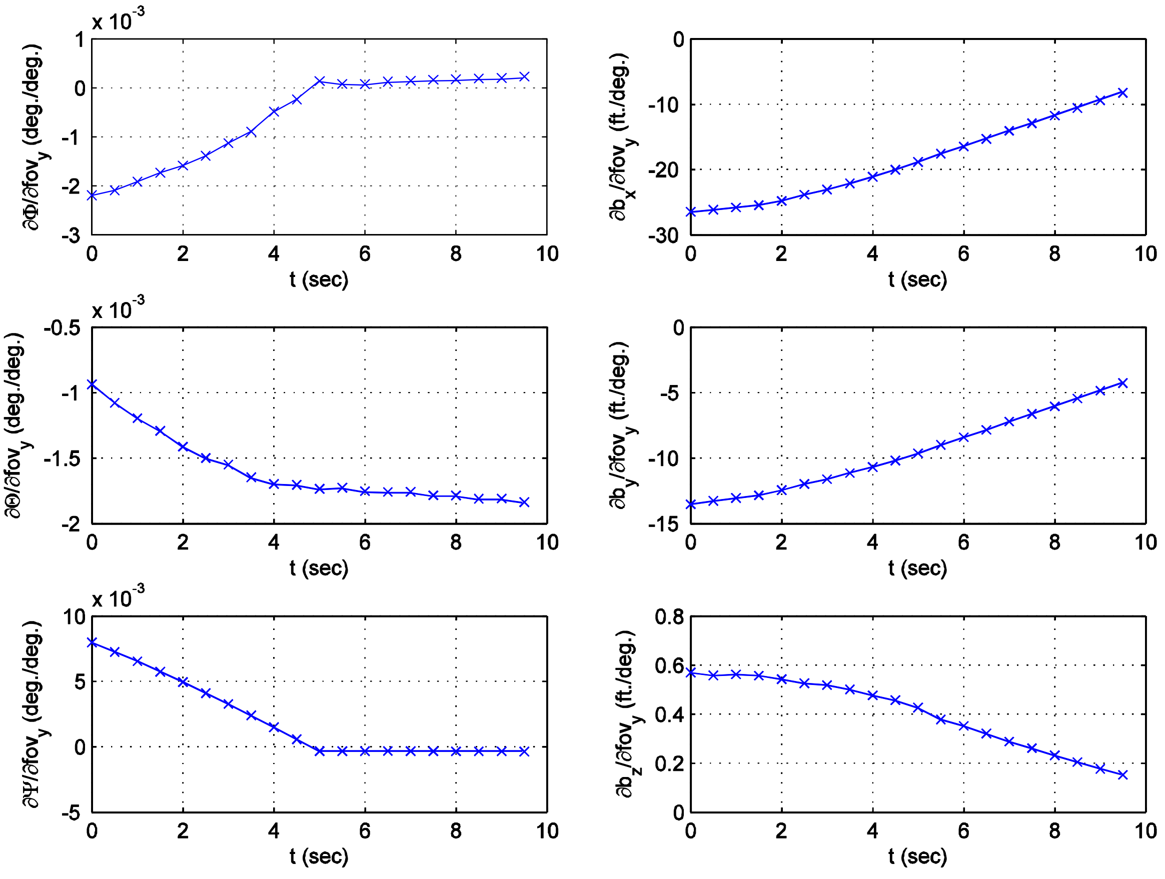


Figure 4.31: Sensitivity of State Estimate to Field-of-View Measurement Error in Synthetic Test 3

4.3.3 Natural Image Tests

Figure 4.32 shows the sensitivity of the Euler angle measurements to RMS corner measurement error in the natural image test. The most significant state estimate error results from measurement errors in the corners' *y*-coordinates; however, overall the estimates appear to be fairly robust.

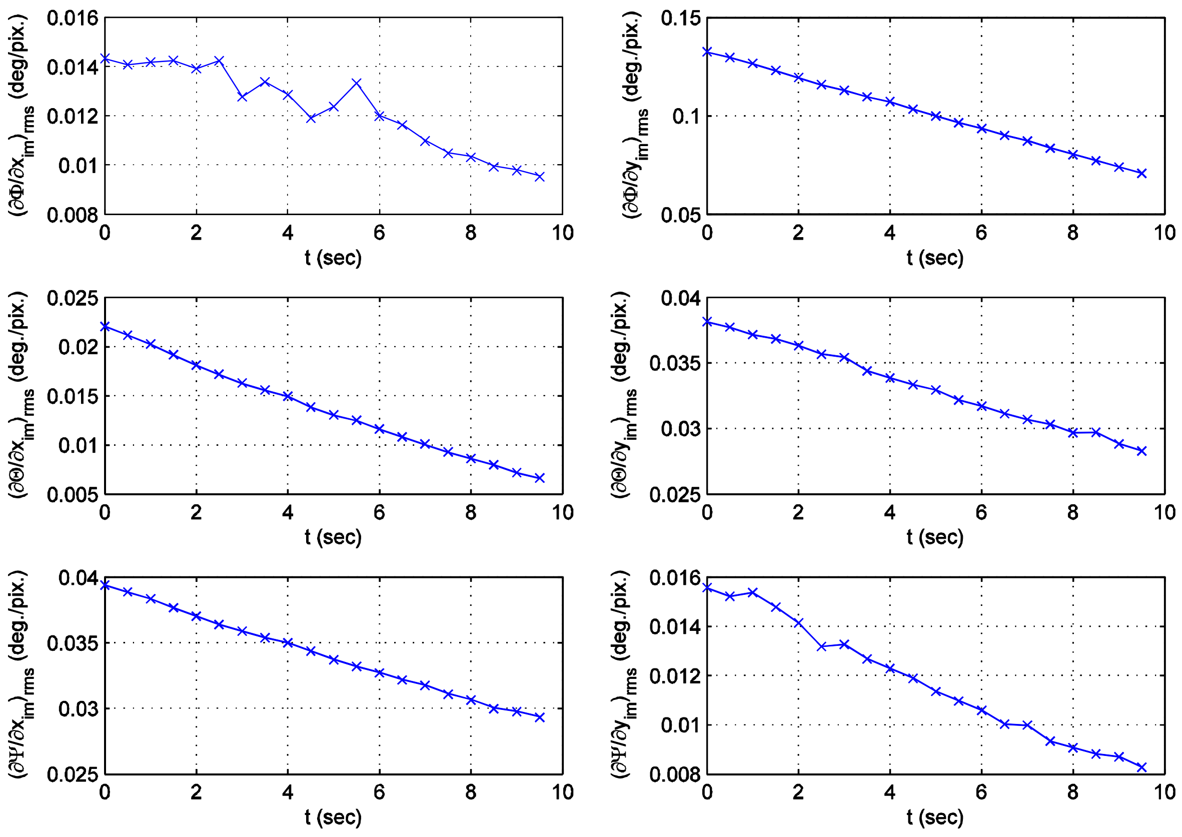


Figure 4.32: Sensitivity of Euler Angle Estimate to RMS Corner Measurement Error in Natural Image Test

Figure 4.33 shows the sensitivity of the position estimates to RMS corner measurement error. Again the algorithm is fairly robust overall, but very significant error in the camera's *x*-coordinate would result from small measurement errors in the corners' image *x*-coordinates.

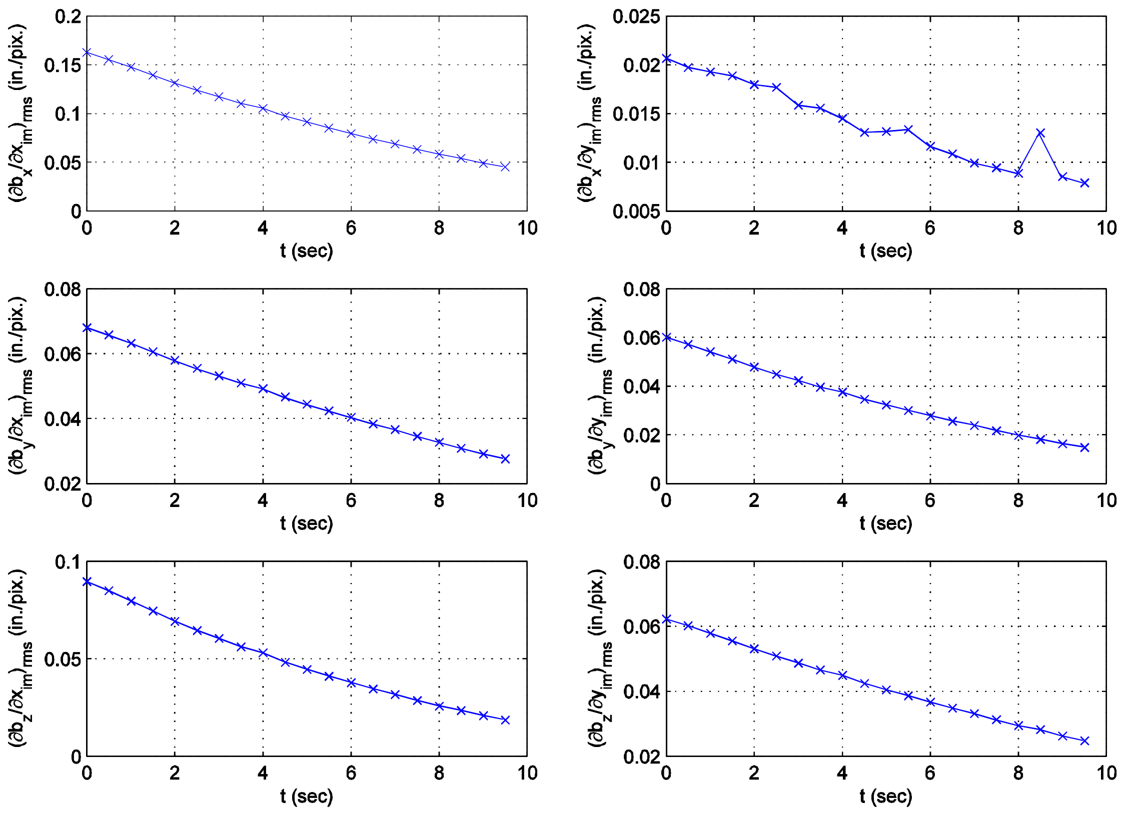


Figure 4.33: Sensitivity of Position Estimate to RMS Corner Measurement Error in Natural Image Test

Figure 4.34 shows the sensitivity of the state estimate to field-of-view measurement error. As is to be expected, Euler angles are relatively insensitive to the field-of-view angle but the distance-to-runway estimation is profoundly affected by the field-of-view angle. This angle also has an impact on the altitude estimate, but its effect diminishes rapidly as the runway comes into larger view.

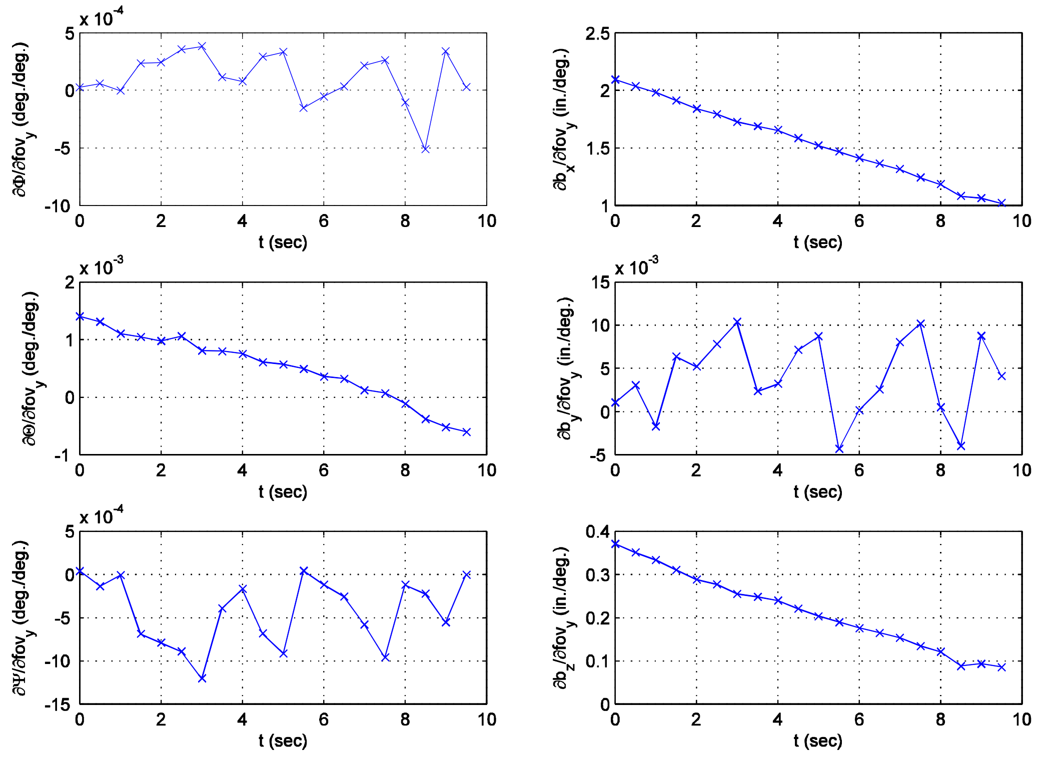


Figure 4.34: Sensitivity of State Estimate to Field-of-View Measurement Error in Natural Image Test

4.4 CPU Load Analysis

4.4.1 Overview

A CPU load analysis is performed for the purposes of identifying bottlenecks and areas for code optimization, should it be desired that this algorithm is implemented in a real-time system. The numbers obtained in this section are combined with the algorithmic complexity analysis from Section 3.2 to project the effect of certain modifications. These results were obtained using an Intel Core i5-2310m at 2.3GHz with 4GB of RAM running Windows 7 64-bit.

4.4.2 Synthetic Image Load Analysis

Figure 4.35 shows the CPU load breakdown for each major sub-algorithm in the system. The Canny Edge detection algorithm dominates overall load on the CPU, making this sub-algorithm the critical bottleneck. Image Acquisition is the second-most-intensive operation, although this number is driven by hard drive access times and would probably not be nearly as significant in a real-time system. SSE and Hough Transform operations are next on the list, although these are both highly parallelizable and could easily be implemented on dedicated hardware.

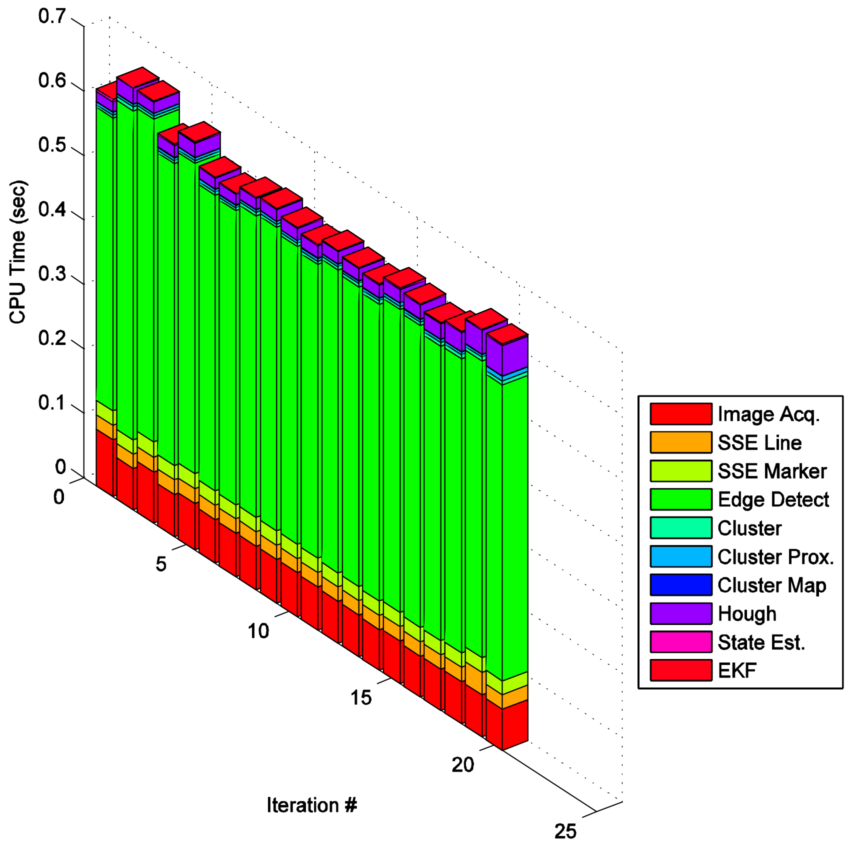


Figure 4.35: CPU Usage Breakdown for Synthetic Test 3

A more detailed breakdown of the CPU load is shown in Figure 4.36. It is worth noting that the Hough transform takes longer to execute as the runway comes into larger and more edge pixels appear in the screen, demonstrating the *O(m*n)* behavior mentioned in Section 3.2.

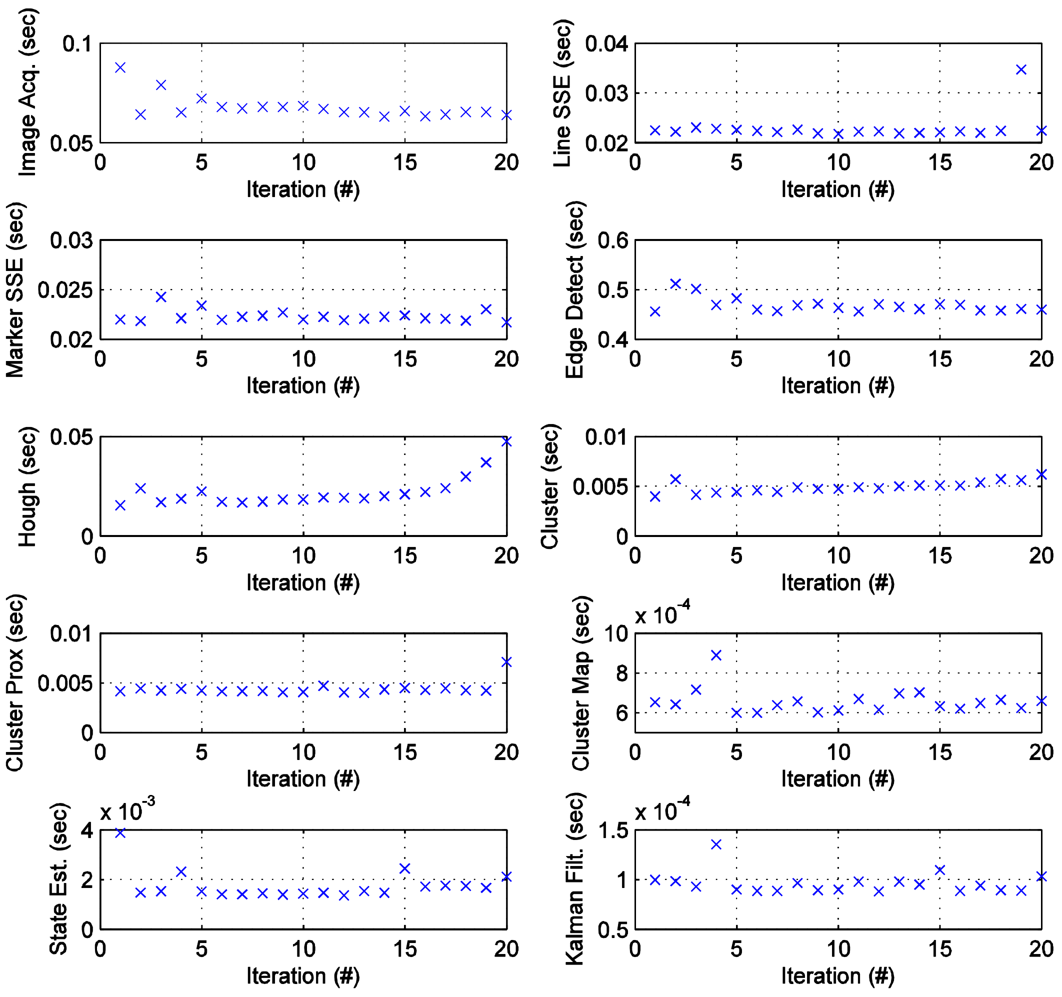


Figure 4.36: CPU Usage History for Synthetic Test 3

4.4.3 Natural Image Load Analysis

The natural image test performance shown in Figure 4.37 has similar characteristics to the synthetic tests, with the exception that the clustering algorithm takes much longer to execute. This is due to the fact that the clustering algorithm is recursively defined, and there are many more marker pixels to process in the natural images than in the synthetic images.

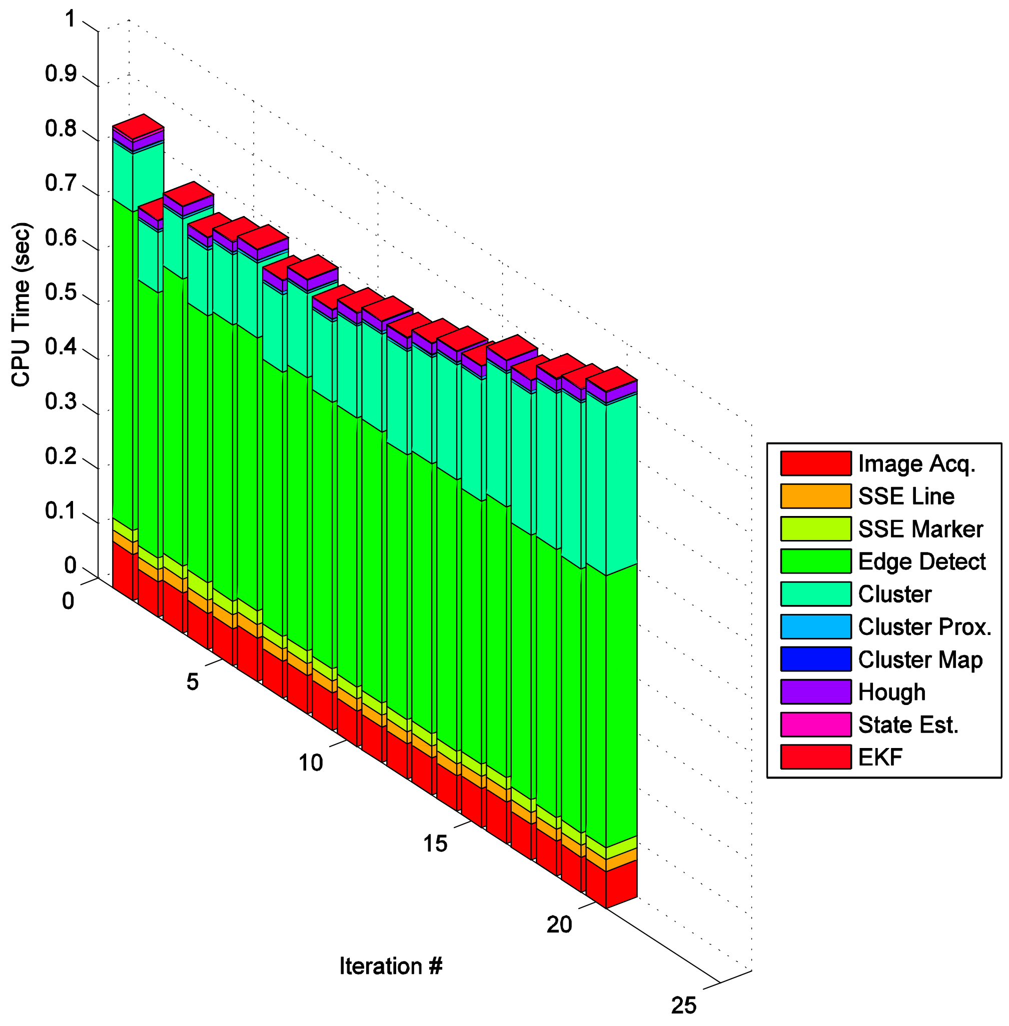


Figure 4.37: CPU Usage Breakdown for Natural Image Data Set

A more detailed view of the CPU load history is shown in Figure 4.38.

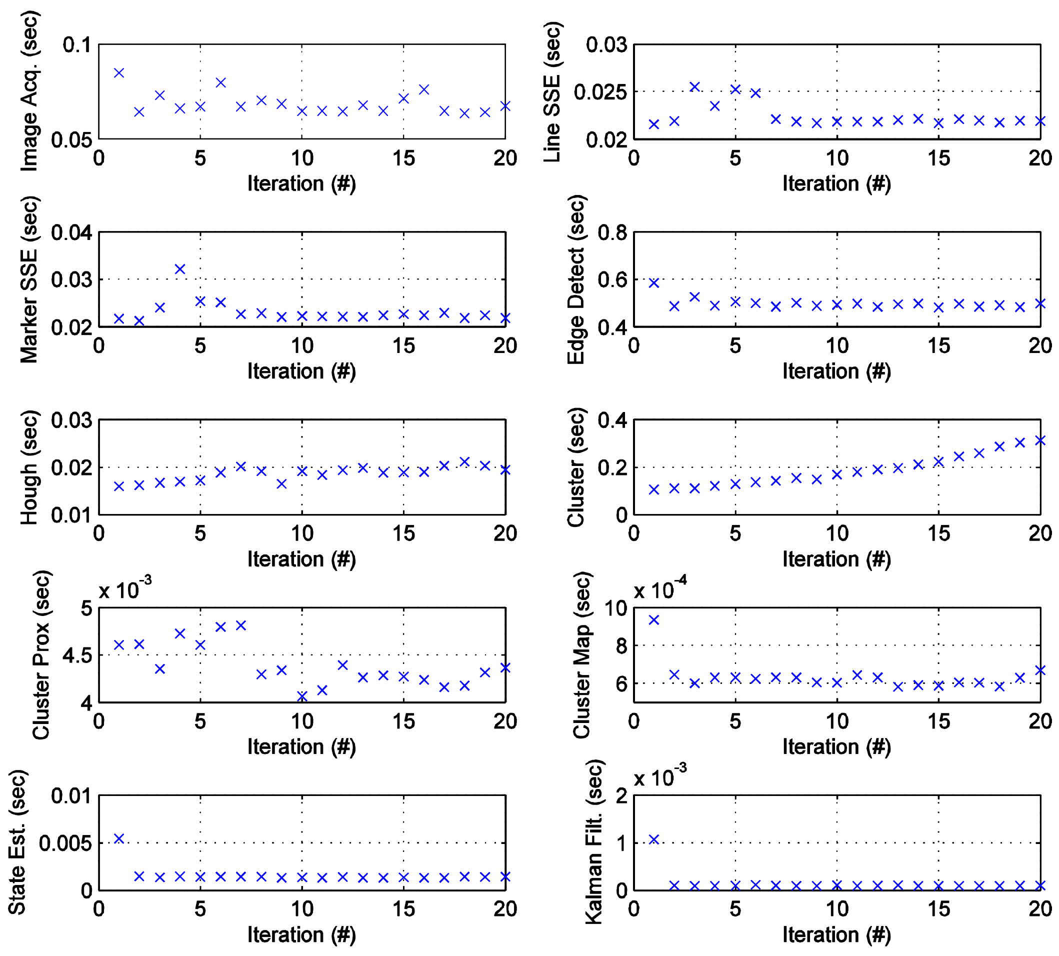


Figure 4.38: CPU Usage History for Natural Image Data Set

4.5 Impact of Using Other Resolutions on CPU Load

The impact on CPU usage time from change in image resolution was estimated using the algorithmic complexity analysis summarized in Table 3.4 and cherrypicked load data from a typical iteration in the natural image test (typically iteration #3). Figure 4.39 shows the effect on CPU load time of varying the image resolution. The algorithm in general scales only slightly worse than linearly with the number of pixels in the image. The following summarizes the results:

- 20 Hz is achievable at 256x192

- 5 Hz is achievable at 512x384
- 1 Hz is achievable at 1024x768
- 0.25 Hz is achievable at 2048x768

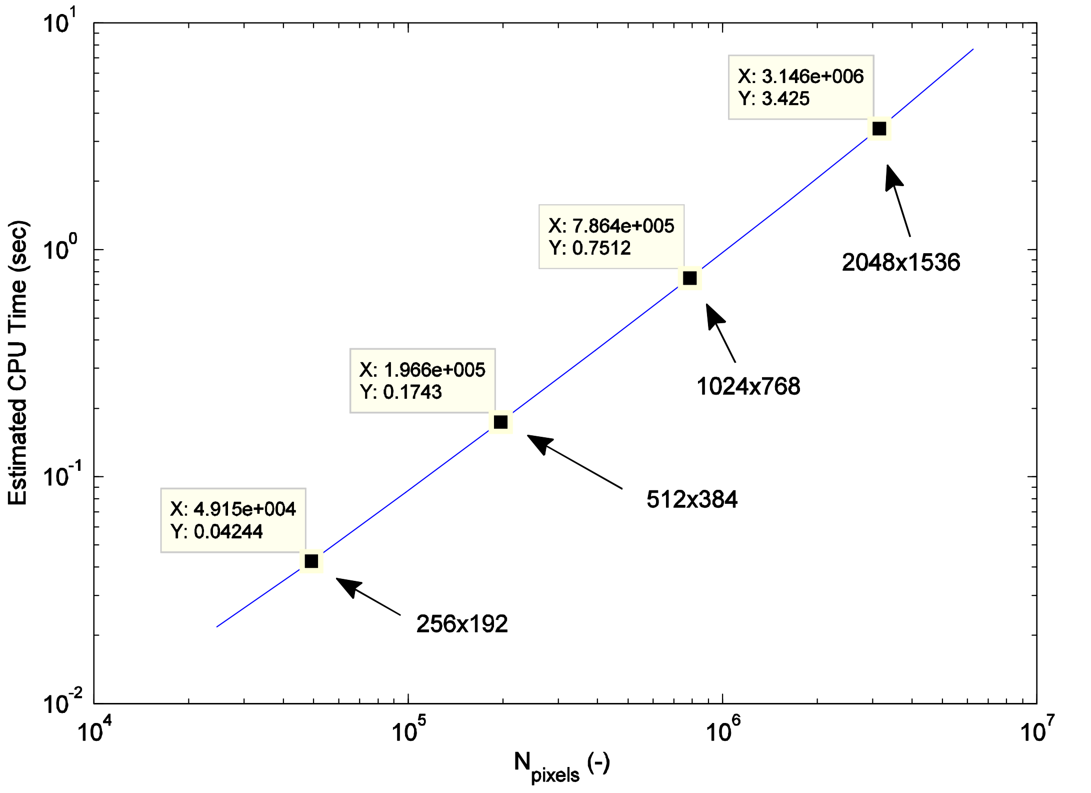


Figure 4.39: CPU Load Scaling from Variation in Camera Resolution

Using lower resolutions would introduce the following problems:

- Markers may not be detected properly
- Runway edges may not be detected properly
- In the worst case, accuracy of the solution would degrade by at least 1 pixel for each halving of image resolution, as per the sensitivity analysis shown in Section 4.3. In the best case, the centroid of each marker would be identical for each image resolution and the accuracy would be unaffected.

A better approach to solving the performance problem than lowering the resolution would be to use more sophisticated hardware. The Canny edge detector alone accounts for approximately 2/3's of overall CPU usage. Recently, a Xilinx Spartan-6 FPGA implementation of this algorithm was developed that can perform this algorithm on a 1280x960 image in 3.09 milliseconds [20] – a 160-fold improvement over the microprocessor-based implementation in this algorithm. The SSE operations could also easily be offloaded to an FPGA since they are highly separable and do not require floating point arithmetic. The other major performance bottleneck is the clustering algorithm. Since this algorithm is recursive and it is implemented in MATLAB, it is likely that MATLAB's interpreter is the source of the inefficiency. It is reasonable to assume that if the algorithm were rewritten in C++, it would execute much more quickly. If all of these steps were taken, it is certain that this algorithm could be implemented in real-time using fast microprocessors and FPGAs.

5 Conclusions and Recommendations

5.1 Conclusions

A vision-based method of UAV state estimation for the purposes of augmenting and backing up GPS-based systems has been developed and presented in this Thesis. This method can be used for fixed-wing conventional landing, rotary wing helipad landing, or fixed-wing net recovery. Additionally, it could be adapted for optionally-piloted vehicles, pilot training devices, or even other robotic platforms. The requirements for the usage of this system are good lighting conditions, clear edges of the recovery target, unicolor markers on the recovery target's corners, and knowledge of the recovery target's physical dimensions and location.

Accuracy of this method has been demonstrated in synthetic images on a 7500 foot runway to within 20 feet laterally and 4 feet vertically initially, diminishing to nearly 0 as the runway comes into larger view. This level of accuracy is certainly on par with GPS. However, a large portion of the error is very likely due to incorrect assumptions about how the runway is rendered in FlightGear. It is possible that FlightGear uses a different definition for the perimeter of the runway (perhaps in the middle of the white line instead of the far edge) – causing significant position error. It must be noted that this sort of error would not be present in a real system if care is taken to use consistent conventions. Accuracy of pitch and roll was consistent to within 0.1 degrees – far better than what any other method achieved in the literature review. Heading had a consistent bias of approximately -0.75 degrees, which is likely caused by an "off-by-1" indexing error somewhere in the implementation code. This could be occurring in "convert_pixels_to_ndc.m" or its related inverse operations, which convert image coordinates to and from normalized device coordinates. A constant error in this conversion would certainly cause a consistent bias in heading angle.

Accuracy of the algorithm has been demonstrated on a set of natural images on a 90 inch runway to within 0.5 inches laterally and 0.4 inches vertically. Again, the error in altitude exhibits the favorable behavior of approaching 0 as the runway comes into larger view. Additionally, with small glideslope angles the altitude sensitivity is much lower than the lateral position sensitivity. This is a favorable property for traditional fixed-wing recovery and is a natural result of the projection process. The errors that do exist in the position estimate, though quite small, can be attributed to systematic error involved with exhibiting asymmetric forces on the camera while taking pictures using the track apparatus. Error in Euler angles is somewhat larger than in the synthetic tests, reaching up to 1.2 degrees in roll and up to 0.6 degrees in pitch. Again, heading angle exhibits a slightly negative bias.

If position accuracy is normalized to runway length, then it is possible to get a feel for the scaling effects of the algorithm. The results are summarized in Table 5.1. Note well that these numbers are for the very worst individual observation (note optimal estimate) in each test run. Despite this, even the worst observations yield very good relative results. The way to interpret this is as follows: in each test, the algorithm never gave an observation worse than 0.48% of the total length of the runway in lateral position or 0.44% of the total length of the runway in vertical position. In fact, as the camera approached the runway, observations typically got much better and the Kalman Filter greatly improved tracking of the system.

Table 5.1: Summary of Worst-Case Position Observation Accuracy Normalized With Runway Length

	Synthetic 1	**Synthetic 2**	**Synthetic 3**	**Natural**
$\delta b_{x\text{-}max}/l$ (-):	0.2%	0.2%	0.24%	0.39%
$\delta b_{y\text{-}max}/l$ (-):	0.12%	0.14%	0.13%	0.28%
$\delta b_{z\text{-}max}/l$ (-):	0.05%	0.03%	0.05%	0.44%

If the natural image accuracy numbers were applied to a more typical UAV runway length of about 1000 feet and with all other things being equal, then position accuracy could be expected to be within 5 feet in the worst case and within a few inches as the vehicle approaches the threshold. With more precise markers, slight modifications to marker centroid detection, and better knowledge of the true field-of-view angle even better position estimation could be realized. This level of position knowledge is certainly sufficient for UAV autolanding.

The algorithm in its current implementation suffers from two major deficiencies:

1. Color information is relied upon to detect recovery target edges and markers. Lighting conditions and camera calibration will significantly impact the algorithm's ability to detect both of these key features. This could be solved by categorizing pixels according to their hue as opposed to sum-square error from a specific color, putting these values into a histogram, and then thresholding the image such that at least a certain number of pixels are always flagged. These pixels could then be searched for neighbors with similar color, thus allowing for some tolerance for shading and sensor noise. These clusters would then passed into the clustering algorithm and the rest of the system would iterate as normal.
2. The algorithm will fail once the recovery target moves offscreen of the image. This could be solved by placing intermediate markers inside the recovery target:

- For a runway, two new markers could be placed halfway between the opposing thresholds. Since aircraft typically land in the first 10-30% of the runway, this should allow the system sufficient margin to always have at least 4 markers in view.
- For a helipad, an additional rectangle could be placed near the center of the pad that would be sufficiently small to never completely leave the view of the camera on landing.
- For a net, this is likely not a problem since the vehicle will be essentially within the capture zone of the net once the corners leave the view of the camera.

An additional layer of intelligence would need to be added to the algorithm to allow for markers to leave the view of the camera. This would most easily be accomplished by modifying the 2D point to 3D point optimizer as follows:

1. First find the optimal mapping for the furthest 4 points (the inner rectangle in the case of the helipad, or the far four points in the case of the runway).
2. Holding this mapping constant, run the optimizer for each combination of N-4 points needed to complete the rest of the mapping. If there are insufficient detected clusters to do this, then stop the algorithm and proceed with the results from the previous step.
3. If the average resulting cost per point (i.e., average sum-square distance error per point) for the optimal map rises significantly from the value obtained in the first step, then the mapping can be considered invalid and the results from step 1 should be used.

Finally, the choice of marker geometry was suboptimal. The cross-section of a flat ellipse obviously diminishes to 0 as the aspect angle goes to 0, which is precisely what happens during a shallow landing. It is desirable to choose a geometry that has a 2D centroid that coincides with a 3D center of mass – spheres would be much better choices.

5.2 Recommendations

It is recommended as a first step that this system be further tested due to its necessity for providing additional navigation robustness to the next generation of civil and military aircraft and its exceptionally good performance relative to previously-developed systems. It is recommended to implement a spherical marker system on an available runway, mount a high-definition camera onto a GA aircraft, calibrate it, and perform a flight test of the system. The flight test should include flyovers as well as ghost landings. The merit of placing markers on locations other than the far threshold of the runway should also be investigated. On some very long runways, it may be difficult for the vision system to detect the markers due to haze or heat distortion. The far-side markers could be placed much closer, and this could improve not just marker detection but estimation accuracy.

If reasonable state estimates are obtained from the flight test video, then it is recommended to use this algorithm as a means of performing system identification. The Euler angle estimates and position estimates are far better than what would be expected of typical Inertial Navigation Systems. If integrated with vehicle control inputs, it is anticipated that very high-fidelity linear models could be created using the data from this system.

If this program is successful, it is recommended to add the intelligence necessary to provide additional robustness against color perception error as well as markers leaving the camera's field of view. These updates to the algorithm should also be thoroughly flight tested.

If this step is successful, then the algorithm could be implemented in a real-time system. The core algorithm would need to be converted from MATLAB code to faster C++ code, probably using the OpenCV library. A fast multicore embedded processor would need to be selected for use, and separable algorithms such as the Hough Transform be split amongst the various cores. The Canny Edge Detector and SSE calculations should be offloaded to an FPGA, since these operations can be done much faster in highly-parallel devices than on a microprocessor. In addition to the required work on the core state estimator, an actual autoland controller designed for use with this system would also need to be developed.

While such an undertaking would require a very significant amount of engineering, it is worthwhile if its performance lives up to its promise. A unified vision-based method for recovering a UAV of any type or size regardless of whether the target is a runway, helipad, or net would present a massive leap forward in the state-of-the-art for computer vision and navigation technology, and vastly improve the robustness of next-generation aerial autonomous platforms.

6 References

[1] National Space-Based Positioning, Navigation, and Timing Advisory Board. Jamming the Global Positioning System – A National Security Threat: Recent Events and Potential Cures. http://www.pnt.gov. November 4, 2010.

[2] D. Hambling. GPS Chaos: How a $30 Box Can Jam Your Life. New Scientist, Issue 2803. March 6, 2011.

[3] MEMSIC. NAV420CA Datasheet. http://www.memsic.com. Retrieved June 15, 2011.

[4] Xsens. MTi-G Datasheet. http://www.xsens.com. Retrieved June 15, 2011.

[5] C. Sharp, O. Shakernia, and S. Sastry. A Vision System for Landing an Unmanned Aerial Vehicle. IEEE International Conference on Robotics and Automation, pp. 1720-1727 Vol. 2, 2001.

[6] S. Saripalli, J. Montgomery, G. Sukhatme. Vision-based Autonomous Landing of an Unmanned Aerial Vehicle. IEEE International Conference on Robotics and Automation. pp. 2799-2804, 2002.

[7] O. Shakernia, Y. Ma, T. J. Koo, S. Sastry, Landing an Unmanned Aerial Vehicle: Vision Based Motion Estimation and Nonlinear Control. Asian Journal of Control, Vol. 1, No. 3, pp. 128-145, 1999.

[8] T. Templeton, D. Shim, C. Geyer, S. Sastry. Autonomous Vision-based Landing and Terrain Mapping Using an MPC-controlled Unmanned Rotorcraft. IEEE International Conference on Robotics and Automation, pp. 1349-1356, 2007.

[9] H. Wang, J. Peng, L. Li. Runway Detecting and Tracking of an Unmanned Aerial Landing Vehicle Based on Vision. International Journal of Pattern Recognition. Vol. 20, No. 8, pp. 1225-1244. 2006.

[10]M. Betke, L. Gurvits. Mobile Robot Localization Using Landmarks. IEEE Transactions on Robotics and Automation. Vol. 13, No. 2, pp. 251-263, 1997.

[11]A. Cesetti, E. Frontoni, A. Mancini, P. Zingaretti, S. Longhi. A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks. Journal of Intelligent and Robotic Systems. Volume 57, No. 1-4, pp. 233-257.

[12]E. Frew, J. Langelaan, S. Joo. Adaptive Receding Horizon Control for Vision-Based Navigation of Small Unmanned Aircraft. American Controls Conference, 2006. pp. 2160-2165.

[13]J. Canny. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 8, No. 6, pp. 679-698.

[14]P.V.C. Hough. Machine Analysis of Bubble Chamber Pictures. International Conference on High Energy Accelerators and Instrumentation, 1959.

[15]J. Roskam. Airplane Flight Dynamics and Automatic Flight Controls. Part I, pp. 15-20. DARcorporation, Lawrence, KS, 2007.

[16]D. Shreiner. OpenGL Programming Guide, Chaper 3: Viewing. Addison-Wesley Professional, Reading, MA, 2009.

[17]S. Buss. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods. Department of Mathematics, University of California at San Diego, Oct. 7, 2009. http://math.ucsd.edu/~sbuss/ResearchWeb/ikmethods/iksurvey.pdf.

[18]R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, Vol. 82, Series D, pp. 35-45. 1960.

[19]R.F. Stengel. Optimal Control and Estimation. 2nd Edition. Dover Publications, Mineola, NY. pp. 342-345.

[20]FlightGear Development Team, FlightGear v. 2.0.0. http://www.flightgear.org.

[21]C. Gentsos, C. Sotiropoulou, S. Nikolaidis. Real- Time Canny Edge Detection Parallel Implementation for FPGAs. IEEE 17th International Conference on Electronics, Circuits, and Systems. pp. 499-502.