Winter 2014

DEVELOPMENT OF THE NASA CELESTIAL NAVIGATION METHOD FOR DYNAMIC EXTRATERRESTRIAL SURFACE NAVIGATION

Jared Carl Perkins
University of New Hampshire, Durham

DEVELOPMENT OF THE NASA CELESTIAL NAVIGATION METHOD FOR DYNAMIC EXTRATERRESTRIAL SURFACE NAVIGATION

BY

Jared Perkins

B.S., University of New Hampshire, 2008

THESIS

Submitted to the University of New Hampshire

in partial fulfilment of

the requirements for the degree of

Master of Science

in

Mechanical Engineering

December 2014

This thesis had been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering by:

Thesis Director, May-Win L. Thein

Associate Professor of Mechanical Engineering

Barry Fussell

Professor of Mechanical Engineering

Michael Carter

Associate Professor of Electrical and Computer Engineering

On December 3, 2014

Original approval signatures are on file with the University of New Hampshire Graduate School.

Acknowledgements

I want to give my great thanks to all those who have been supportive these last ten years through my undergraduate and graduate career. Mostly I want to thank my wife, Deborah, for all her great support and encouraging me to keep pushing to the end.

I want to thank Professor May-Win Thein for giving me the the opportunity to work with NASA as a Research Assistant.

Also my thanks goes to Professor Barry Fussell and Michael Carter for being on my thesis defense committee.

I want to thank all at UNH, my friends and family for all their love and support, and for a great experience at UNH!

Table of Contents

List of Figures

List of Tables

Nomenclature

Γ	- Accelerometer Transformation Matrix
Δ	- Star Tracker Transformation Matrix
M_A	- Accelerometer Alignment Matrix
M_C	- Star Tracker Alignment Matrix
Ω	- Planetary Gravity Model(Matrix)
Φ_A	- Celestial Body
ϵ	- Heading
λ	- Latitude
ϕ	- Longitude
α	- Tilt
β	- Slope
ENU	- East-North-Up Reference Frame
NED	- North-East-Down Reference Frame
SD	- Selenodetic Reference Frame
SC	- Selenocentric Reference Frame
θ	- Rotation Angle of Rover/Astronaut
μ	- Mean
σ	- Standard Deviation

Abstract

DEVELOPMENT OF THE NASA CELESTIAL NAVIGATION(CelNav) METHOD FOR DYNAMIC EXTRATERRESTRIAL SURFACE NAVIGATION

by

Jared Perkins

University of New Hampshire, December 2014

The Celestial Navigation (CelNav) method was developed in conjunction with NASA Goddard Space Flight Center, to provide accurate location data for extraterrestrial surface navigation without the use of a global positioning system (GPS) or a ground/relay station. CelNav is a minimal sensor/power solution originally developed for static Lunar surface navigation. However, dynamic navigation via CelNav requires high-accuracy state estimates, due to the absence of key sensors such as a gyroscope, GPS, and a magnetometer.

In this thesis, robust nonlinear state estimation techniques (the Sliding Mode Observer, the Extended Kalman Filter, and the H-Infinity Filter) are used with CelNav to accurately determine dynamic latitude, longitude, and heading, for an unmanned/manned rover or astronaut. The goal is to investigate the feasibility of implementing a nonlinear estimation technique with CelNav for dynamic extraterrestrial surface navigation when accurate location coordinates are necessary. Preliminary results show that this research shows promise as a secondary dynamic navigation system for future extraterrestrial exploration.

Introduction

1 The NASA Celestial Navigation (CelNav) Method

The NASA Celestial Navigation is a technique to determine a space vehicle (rover) or astronaut's navigational coordinates (latitude and longitude, as well as heading) on an extraterrestrial surface. It was developed as a low-cost secondary system and fault checking device to facilitate greater coordinate location when primary navigation systems no longer function (e.g. solar disruption, absence of communication with ground stations).

This basic idea was proposed in 2006 by Quinn [22], who deduced that with the rudimentary instruments already contained within space vehicles it would be possible, if the error can be assumed to be small, to accurately determine one's location. Using small angle approximation, one can determine that the accuracy needed in all sensors to restrict navigational error to 50m (the original NASA target accuracy level). That is:

$$\frac{0.050km}{R_{Moon}} * \frac{648000\text{arcseconds}}{\pi} = 5.93\text{arcseconds} \tag{1}$$

This fact was later confirmed through the first CelNav algorithm further developed by Thein et al. [2]. Using this information CelNav is further refined and streamlined into the algorithm that currently stands.

1.1 CelNav History

CelNav was originally designed to be used at the Lunar poles in search of pockets of frozen water discovered in shielded craters, such as the Shackelton Crater located on the south pole of the Moon (Figure 1). Future space settlements would depend on this liquid resource. Having the ability to determine accurate location coordinates would be critical to either an astronaut or a space vehicle, especially during loss of communication from either a local base or a ground command center.

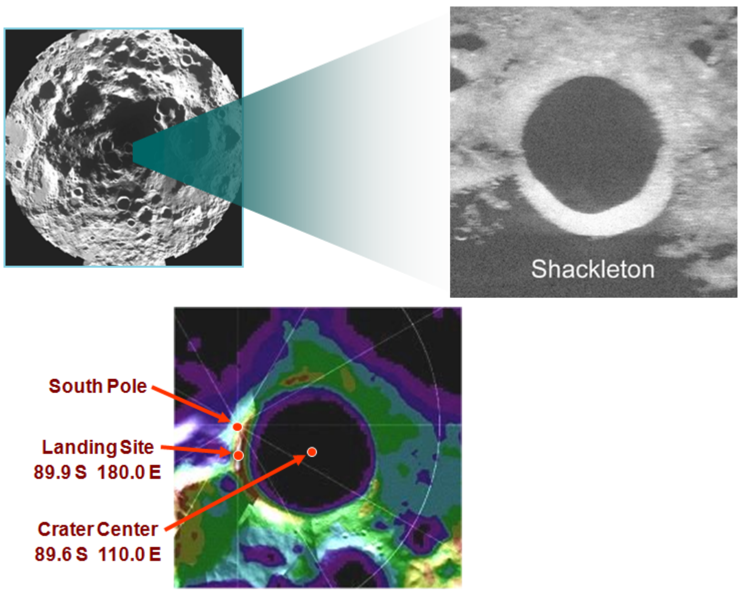

Figure 1: Shackelton Crater located on Earth's Moon [22]

CelNav is not a new concept. Ancient and modern seafarers were able to determine latitude and longitude using celestial navigation with relatively good accuracy. Early nautical navigation was accomplished with few highly advanced mechanical instruments (e.g. sextants, astrolabes, octants, and clocks) to measure the angle between certain celestial bodies such as the Sun, Moon, other planets, or one of the 57 stars contained within a Nautical Almanac [13]. The Nautical Almanac is based on the fact that any heavenly body has a distinct location above the Earth's surface during specific times of the year. This is known as the heavenly body's geographic position (GP) as found in the Nautical Almanac [13].

There are known relations between the angle of the celestial body and the visible horizon, and this information has a direct relation between a GP and the observer's position. From these relations the observer is able to draw a line of position (LOP), and transcribe it on a navigational chart. The observer's position is located on the LOP. This line can be further refined by sighting out another heavenly body, and transposing the new LOP on the navigational chart. The point at which these lines intersect is the observers approximate location. This method is known as "Altitude-Intercept Method" [13].

Altitude-Intercept Method was first attempted using only the human arm as a measurement device to attempt to achieve an accurate angle. Over time advanced mechanical devices were developed to help more accurately measure the angles of heavenly bodies. Of these tools, the sextant (as shown in Figure 2) was developed by Isaac Newton as a highly accurate method for measuring the horizon. A skilled navigator is able to navigate effectively within an error of 1.5 nautical miles. This distance is accurate enough to sight land. This method relies heavily on an accurate chronometer, set to a clock on the Prime Meridian [11].

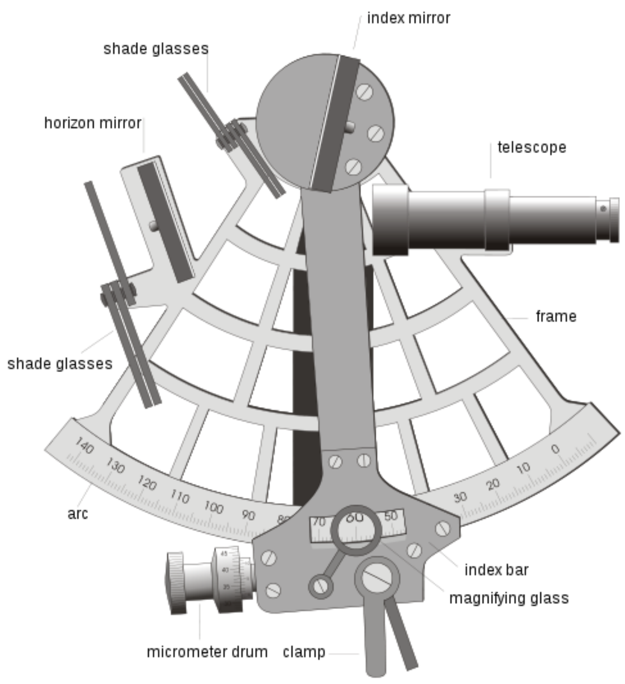

Figure 2: Marine Sextant [20]

CelNav was conceived out of a need for a reliable inertial navigation system (INS) that is self contained and independent of external sensors, such as a Global Positioning System (GPS) type system, as shown in Figure 3, or orbiting satellite(s).

GPS is used heavily by both military and civilians alike to accurately determine the location of a receiver on Earth to within a 10-15 meter radius [12]. This is possible due to a small number (24) of satellites in mid-Earth orbit (approximately 20,000 km altitude). GPS satellites are equipped with an array of sensors that transmit signals to the GPS receiver on Earth that contain [12]:

- the time the message was transmitted
- precise orbital information (the ephemeris)
- general system health and approximate orbits of all GPS satellites

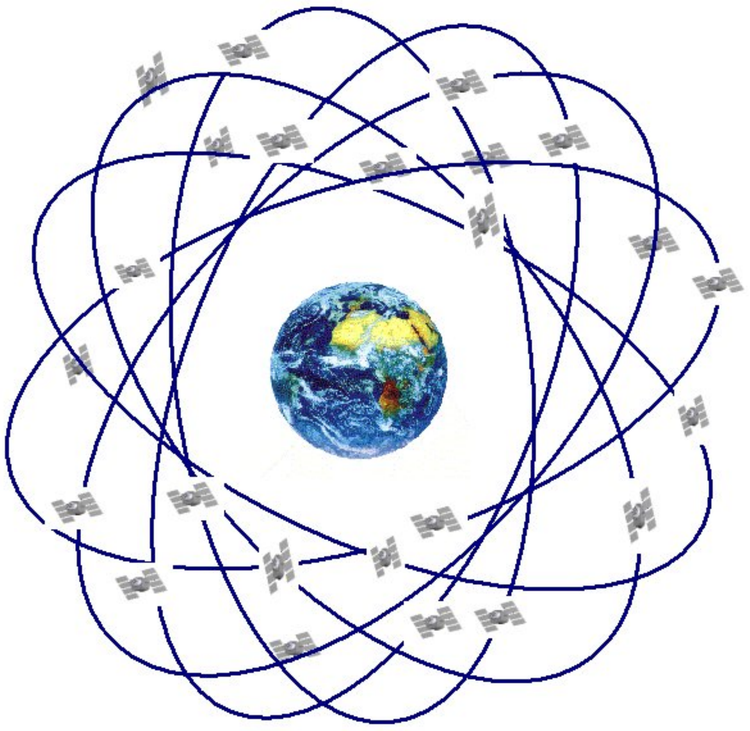

Figure 3: Global Positioning System [21]

Using this information (from a minimum of three satellites) it is possible to obtain latitude, longitude, and altitude. This system is not perfect because it is prone to miscalculations from a multitude of sources including impaired line of sight, imperfect time information, too few satellites, solar activity and aged satellite hardware.

Another method that is used for navigation on extraterrestrial bodies involves sending, at a minimum, a single satellite to orbit the body. This allows for a doppler type positioning system to be used. The doppler position system works on the premise that there is a change in wave (signal) frequency for an observer moving relative to the source of the wave [14]. The frequency received by the observer is higher (when compared to the emitted frequency) during the approach, identical at the instant of "flyover", and lower as the signal moves further away

[14]. The Doppler Effect is useful due to the fact that the orbital information (altitude, velocity, and orbit) is known and carefully monitored. Thus, a receiver on a rover or astronaut is able to obtain navigational information based upon the signal received from the orbiting satellite. This navigation method too, however, is prone to inconsistencies because of, for example, line-of-sight, limits on fuel/power, and solar activity.

Again, these are hindrances that can add uncertainty to an already complicated mission. Thus, a self-contained system, that is not reliant on outside sensors or technology is necessary for the expansion of extraterrestrial exploration. The research presented in this thesis involves developing CelNav for such a purpose.

In the future, long term extraterrestrial missions are expected to have limited power and sensors available. This means that although redundant sensors could be incorporated, there would be no access to additional sensor measurements. These sensors would be limited to star trackers and 3-D accelerometers. These two sensors are used primarily in inertial navigation systems, which output positional navigation data (heading and inertial position). Many of these inertial navigation systems depend on either doppler or local GPS data which, as explained above, would not always be available. These additional systems make it possible to use CelNav to fault check position data.

A star tracker is a CCD (Charge-Coupled Device)-like optical device that compares an image taken by the star tracker to a database of stored images. The output of the device is a unit quaternion (to be explained in greater detail later in this thesis). In general the unit quaternion represents the orientation (heading and position) of an astronaut or rover. The star tracker is oriented orthogonally along the vehicle or astronaut's body reference frame.

3D accelerometers, also called an accelerometer triad, are three orthogonally arranged accelerometers. Orthogonality is described as the orientation of each axis of the accelerometer, x-y-z, in relation to an astronaut's or rover's reference frame.

1.2 On-Board Guidance, Navigation, and Control

The function of CelNav is to act as a "stand alone" navigational determination system and may be applied to any vehicle or astronaut that would traverse an extra-planetary body with the possibility of little to no communication with a ground station. It may also be used if communication is ever lost. Multiple stand alone systems are necessary to ensure self reliance. These systems have multiple redundant sensors for which navigation data may be extracted. On-board guidance and navigation are also quite useful in instances where a rover/astronaut is significantly far away from its ground station because of the time it takes for a signal to travel such a great distance. A communication signal is an electromagnetic wave, a radio wave, which has a fixed amount of time it takes for a signal to travel from one location to another. An example of this is communication from Earth to Mars.

The radio signal travels at the speed of light (about 300000 km/s). Therefore, using the approximation:

$$C = \frac{D}{T} \tag{2}$$

where, D, is the shortest total distance from Earth to Mars, which is 57,936,384km, and C is the speed of light, Equation 2 shows that it requires a minimum of 4 minutes for a signal from Earth to reach Mars and, therefore, at least another 4 minutes to relay any signal back from Mars. In turn, at the furthest point away from Earth a signal from Mars can take upwards of 40 minutes to reach Earth. This is why a great deal of the probes and rovers that are sent into space are designed to be autonomous, since it is not feasible to manually control these vehicles effectively in real-time.

2 Past CelNav Development

The original CelNav problem statement as stated in Thein et al. [2], was developed for the exploration of the Lunar surface. A local coordinate system for the Lunar surface is pre-defined as the East-North-Up (ENU) orthogonal triad. It was assumed a priori knowledge of Lunar coordinate transformations such as selenodetic, selenocentric and Moon center-Moon fixed are known. Given needed accuracies, the Lunar lander was to be accurately located to within a 50m radius of its actual location. Originally all measurements were taken in the static case, which means the rover was considered stationary during measurements.

Future exploration to extraterrestrial bodies will most likely include surveying of land, soil, minerals and especially water. Future space settlements would depend on this information. Having the ability to accurately determine position is required during (extra-planetary) exploration because of the possibility of losing communication with a local base or ground/command station while mining or exploring for water or minerals.

The goal of CelNav is to accurately determine the location of the astronaut/rover on the Lunar surface with minimal use of sensors: a 3D accelerometer and a star tracker.

CelNav originally was required to be accurate to within 50 meters of a desired target location. In order to achieve this goal it was determined that the maximum accumulated error for sensors (including measurement noise and misalignment error) could be no greater that 5.97 arcsec, as described in Thein et al. [2]. Later it was determined that on the Lunar surface, a rover or astronaut is expected to be able to see about 2000m away and, therefore, this distance would be an acceptable accuracy target for emergency navigation.

3 Navigational Techniques

A great deal of current research focuses on extraterrestrial navigation by means of access to precise ephemeris data, such as with an Aeronautical Almanac. As stated in Malay et al [15], where a Martian navigation system is examined using an accurate star almanac as well as an extremely accurate clock, it is possible to navigate and extract positional data to within 100m. This almanac is limited, though, and needs updating on a regular basis. For example an Aeronautical Almanac loses accuracy after nine months. At this point a new almanac needs to be calculated. Updating this almanac is acceptable on missions where radio contact is readily available. Otherwise, the star charts become completely inaccurate after one year. CelNav is a more robust method than that relying on an Aeronautical Almanac and can be much more easily adapted to other extraterrestrial bodies.

Another navigation method is orbital tracking, accomplished by making use of orbiting objects such as the Moon. Such a method is a more advanced use of the doppler effect. As stated in Trautner et al [16], "using current Phobos ephemeris data, the position of a Mars lander can be determined with an accuracy of 5km (1-ϕ) with a single night image." It is further stated that the accuracy can be further improved with greater knowledge of the ephemeris data. This method is useful but limited to extraterrestrial bodies with closely orbiting bodies.

4 Research Goals

The current focus of this research is to develop a dynamic navigation system that enables autonomous extraterrestrial surface navigation. State estimators are needed for control feedback for autonomous control. Due to this need, multiple filter/estimation techniques which incorporate the CelNav navigation determination system are to be examined in this thesis. A simple PID controller, (without loss of generality) will be used. Controller techniques will not be examined in this thesis as it is out of the current scope of this research. Further study is provided by Underwood et al. [31].

As stated in the previous section, the original CelNav algorithm ran on artificially generated data. The "sensor" data was obtained by selecting a desired location and calculating the corresponding accelerometer and star tracker data for said location. This data was then corrupted with upwards of 60 arcseconds of noise on all three axes. A problem found in the original CelNav setup, as discussed in Thein et al., is that the system creates one of an infinite number of possible attitude quaternions. The current scope will focus on three different areas of development of CelNav dynamic navigation:

1. CelNav Performance

2. Observer/Filter Development Using CelNav Feedback for Dynamic Navigation

3. Experimental Implementation on test platform

4.1 CelNav Performance

This section focuses on Monte Carlo analysis and preliminary experimental results. Monte Carlo analysis allows for statistical analysis of a system by changing one variable at a time. This was done for a range of latitude and longitude comparing the navigation error at each location.

The preliminary experimental testing was done with the SkyScout, a personal astronomy system commercially available from Celestron. Skyscout contains multiple sensor systems to be described later. This experiment was to show the validity of the CelNav algorithm using real sensor measurements.

4.2 Observer/Filter Development Using CelNav Feedback for Dynamic Navigation

This section looks at the performance of CelNav implemented on a simulated four-wheeled vehicle with front or rear steering. Two separate dynamic models will be examined. The first is a generic four wheeled vehicle with front wheel steering. Then second is a more accurate robust model used in the experimental test case with all wheel drive and tank like steering. A controller is not of interest here due to the fact CelNav only focuses on the accuracy of the observers and not the stability or validity of the controller as defined in Sun et al. [8]. The rover model is taken from Sun et al. [8]. In place of the controller defined in Sun et al.

a simplified PID controller is used instead.

This model is tested in simulation with three separate observer/filter algorithms with CelNav quality “measurements” used as feedback. Here, the feedback is artificially corrupted. The results from this simulation show which observers have the least variance in the estimated signal compared to the true signal. This comparison is slightly different for the experimental version, to be discussed later. This model is also used for the manual version of the rover for the experimental test simulation.

4.3 Experimental Implementation on Test Platform

Expanding on the original CelNav algorithm, it is necessary to make the system dynamic as oppose to static. A rover has been constructed in conjunction with the University of New Hampshire Luna Cats. Testing of this rover is to be conducted independently of this thesis. Further information about the rover can be found in the Appendix C.

The experimental system is used to do a comparative analysis between Sliding Mode Observer (SMO), Extended Kalman Filter (EKF) and H-Infinity (H_∞). This is to test the accuracy of the estimation techniques and not the controller. The goal is to increase the accuracy of the original CelNav algorithm by estimating the desired location and heading, based on the output of the CelNav algorithm. Results are highly dependent on the accuracy of the sensor package, both in measurement noise and sensor misalignment. Dynamics for all of these systems will be discussed in greater detail in later chapters.

This work focuses on proving that the areas discussed above are feasible separately as well as when they are combined, though some combinations may inevitably work better than others.

5 Outline

The following is a brief description of the chapter contents:

- Chapter 1, Review of coordinate reference frames in relation to the moon and local frames, as well as a review of Euler angle representations and attitude transformations.
- Chapter 2, Review of the CelNav Mobility solution algorithm, as well as corrections made to original algorithm.
- Chapter 3, Discussion of various error sources, including measurement noise, misalignment, and bias for both the accelerometer and the star trackers.
- Chapter 4, Background on Monte Carlo history, choice of Monte Carlo statistical parameters, and analysis of results.
- Chapter 5, Preliminary experimental validation of CelNav focuses on the first experimental tests using the Sky Scout as test sensor.
- Chapter 6, Overviews of analytical simulations focusing on both the nominal and real rover models, as well as the controller and necessary assumptions.
- Chapter 7, Discussion focusing on relating the three estimation techniques as well as listing pros and cons about using each type for the task of CelNav.
- Chapter 8, Summary and comparison of results of all estimation techniques. Also possible future direction this project can take.

Chapter I

Coordinate Reference Frames

This chapter reviews inertial and body-centered reference frames and coordinate transformations. Euler angles and quaternions, as methods of defining attitudes are also reviewed. Without loss of generality the extraterrestrial surface used in this research is the Moon. Any generic lunar/planetary surface may be used.

1.1 Local and Lunar Reference Frames

The two major coordinate systems for local and global reference frames are Moon-Centered-Moon Fixed (MCMF), Figure 1.1 [22], and East-North-Up (ENU), Figure 1.2 [22]. The coordinate systems in this research are defined such that:

$$\left|\begin{array}{c} x \\ y \\ z \end{array}\right| \longrightarrow \text{Body-Centered Reference Frame}$$

$$\left|\begin{array}{c} I \\ II \\ III \end{array}\right| \longrightarrow \text{Selenodetic Reference Frame}$$

$$\left|\begin{array}{l} E \text{ - East} \\ N \text{ - North} \\ U \text{ - Up} \end{array}\right| \longrightarrow \text{Local Reference Frame}$$

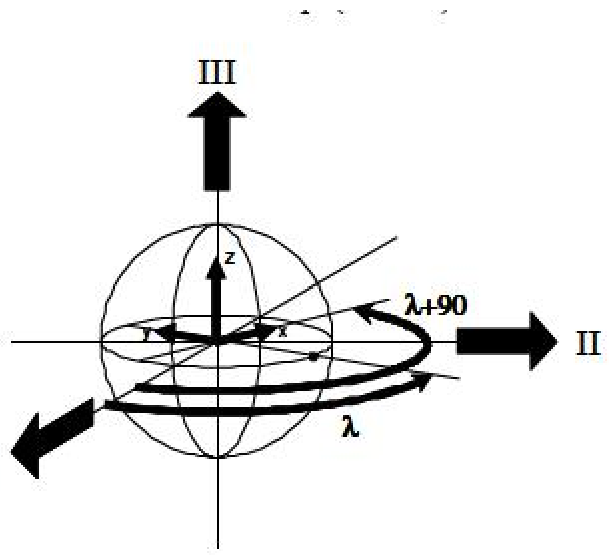

Figure 1.1: Moon Centered Moon Fixed Reference Frame Shown (I-II-III Triad)

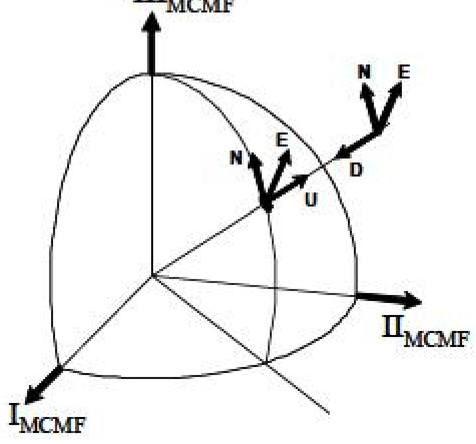

Figure 1.2: ENU Reference Frame

A “Down” reference frame is also used. “Down” can be conveniently used to represent the ENU coordinate system as the “negative Up” orientation and with a generic heading ϵ (i.e., as in the resulting output of an accelerometer). (The ENU reference frame is commonly used by aviators in navigation.)

1.2 Attitude Representation - Euler Angles and Quaternions

The two traditional methods for defining the attitude (orientation) of an object in three dimensional space are Euler angles and quaternions. These methods will be discussed here, along with their respective properties.

1.2.1 Euler Angles

Euler angles are a way to describe any generic rotation as a pre-defined sequence of three pure rotations(i.e about the x, y, and z body-centered coordinate axes). The corresponding angles of rotation are denoted as ϕ, θ, ψ. One example is a 3-1-3 rotation sequence (as shown in Figure 1.3), where:

1. A rotation about the inertial Z-axis by and angle ϕ.
2. A rotation about the new X-axis(x_1) by and angle θ.
3. A rotation about the new Z-axis(z_2) by and angle ψ.

The 3-1-3 rotation sequence is one of 12 possible sequences. (A list can be found in Appendix F.)

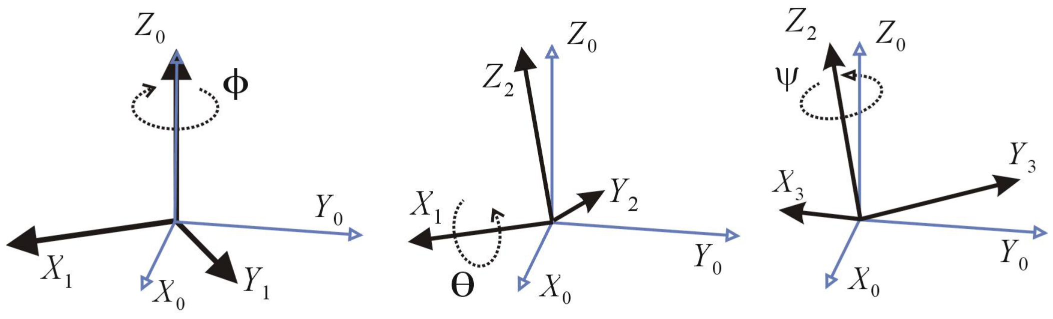

Figure 1.3: Euler Angle Rotations [27]

1.2.2 Direction Cosine Matrix

A Direction Cosine Matrix (DCM)combines an Euler rotation sequence made up of individual rotation matrices into an equivalent single 3 x 3 rotation matrix.[25] This is done through the multiplication of the three rotation matrices. For example the individual DCM rotation matrices that describes a 3-2-1 rotation sequence are:

$$\text{Rotation 1} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.1}$$

$$\text{Rotation 2} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{1.2}$$

$$\text{Rotation 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{1.3}$$

A complete listing of all DCM Euler angles can be found in Appendix F. [25] The resulting DCM for a 3-2-1 rotation sequence is:

$$\begin{bmatrix} \cos(\phi)\cos(\theta) & \sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\sin(\psi) - \cos(\phi)\sin(\theta)\cos(\psi) \\ -\sin(\phi)\cos(\theta) & \cos(\phi)\cos(\psi) - \sin(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) \\ \sin(\theta) & -\cos(\theta)\sin(\psi) & \cos(\theta)\cos(\psi) \end{bmatrix} \tag{1.4}$$

The individual rotations in an Euler angle rotation sequence is physically realizable. As seen in Figure 1.4, the x, y, and z-axis rotations are often referred to respectively as “roll”, “pitch”, and “yaw.”

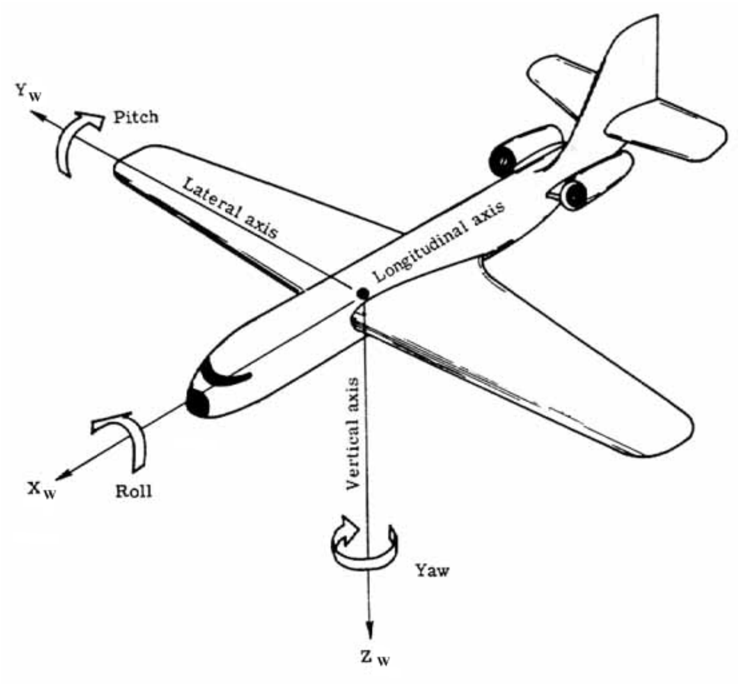

Figure 1.4: “Roll”,“Pitch”, and “Yaw” [28]

Although physically realizable, singularities in the attitude kinematics can occur. A gyroscope is used to illustrate this phenomenon. In Figure 1.5 a gimbal is a platform(or ring) that is free to rotate about a single axis. For the case of aeronautics applications three gimbals are used. This configuration of the gimbals is often used as an Inertial Measurement Unit (IMU). By mounting the set of gimbals to the center of the local body-reference frame, one is able to determine the system orientation with respect to an inertial reference frame. A problem occurs, however, when any two gimbals are aligned in the same plane thus reducing the movement of the gimbals to less than three degrees of freedom.[8] This phenomenon is referred to as gimbal lock. A mathematical equivalent of this gimbal lock may also be shown e.g. 3-1-3 rotation sequence and θ=0; resulting 3-1-3 attitude matrix only have two rotations.

$$\begin{bmatrix} -\sin(\phi)\cos(\theta)\sin(\psi)+\cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta)\cos(\psi)+\cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta) \\ -\cos(\phi)\cos(\theta)\sin(\psi)-\sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta)\cos(\psi)-\sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta) \\ \sin(\theta)\sin(\psi) & -\sin(\theta)\cos(\psi) & \cos(\theta) \end{bmatrix} \quad (1.5)$$

Gimbal lock can be corrected in a number of ways, such as adding an additional gimbal. Another way is to physically move one of the locked gimbals to free the motion. Alternatively, quaternion attitude representation maybe used instead of Euler angles to avoid any such singularities.

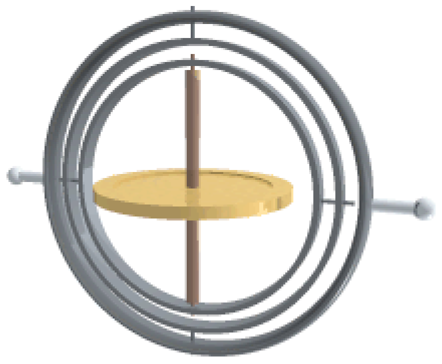
Figure 1.5: Gimbal Lock [29]

1.2.3 Quaternions

The rotations described in the previous section can also be represented in quaternions. A theorem by Euler states:

> Any two independent orthogonal coordinate frames can be related by a sequence of rotations (not more than three) about coordinate axes, where no two successive rotations may be about the same axis.[25]

The basic form a quaternion takes can be seen in Equation 1.6.

$$q = \left[\begin{array}{c} q_0 \\ iq_1 \\ jq_2 \\ kq_3 \end{array} \right| \tag{1.6}$$

or,

$$q = \begin{bmatrix} q_0 \\ q \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\frac{\Phi}{2}) \\ e_1 \sin(\frac{\Phi}{2}) \\ e_2 \sin(\frac{\Phi}{2}) \\ e_3 \sin(\frac{\Phi}{2}) \end{bmatrix} \tag{1.7}$$

A quaternion is a tensor comprised of two distinct parts: a scalar,q_0 , and a vector,$[q_1\ q_2\ q_3]^T$. Also shown in Equation 1.7 is the rotation axis e and the rotation angle Φ.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{1.8}$$

The terms of the tensor are such that they obey the following normalization:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \tag{1.9}$$

This proof is necessary such that if the solution does not equal 1 then the quaternion is not a valid rotation. A major visual difference, as well as why gimbal lock does not exist, is that a quaternion rotation is a single rotation, where as Euler rotations are a combination of a series of separate rotations.

The equivalent DCM for a quaternion given in Equation 1.6 and 1.7 is:

$$A = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2 \\ 2(q_1q_2 - q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2 & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{1.10}$$

As shown in Equation 1.10, the major difference between the Euler angle DCM and that of a quaternion is that the quaternion DCM has no trigonometric functions. It is important to note that both the Euler angle rotation matrix and the quaternion rotation matrix generate the same result for the same location. Thus, there is an innate advantage to using the less computationally intensive method for determining a rotation matrix.[26]

Both the Euler angle and quaternion attitude representations are used in the development of CelNav and in the various estimation techniques in this research.

Chapter II

Celestial Navigation

2.1 Assumptions

The first major assumption that is made is the determination of a planetary body's gravity model described as $[\Omega]_{SD}^{Inerital}$. The model used in this research is based on the Lunar surface. For simplification, and because the Moon is almost a perfect symmetrical body, we assume, without loss of generality, $[\Omega]_{SD}^{Inerital} = 1$;

The second assumption was discovered during numerous statistical simulations (i.e. Monte Carlo Analysis); almost all of the sensor error comes from the misalignment of the accelerometer.

2.2 Development

Celestial Navigation (CelNav) grew out of a need for accurate, low cost and low power position determination solution for extra planetary bodies. CelNav was originally developed as a NASA sub-project of the Robotics Lunar Exploration Program (RLEP). This project's original mission was to explore craters in the

South Pole region of the Moon for frozen water. The original CelNav algorithm was based on work done by Swanzy et al. [1] Compass Star Tracker (CST). However, the local nadir is not accounted for, which is affected by surface terrain (e.g, craters, rocks,etc). This research assumed that the star tracker is always aligned with the local zenith, unrestricted by terrain and always laying upright on a level surface. [2]

The CelNav algorithm was originally developed in 2007 by Thein et al. The algorithm was further refined and updated in 2008 to account for greater measurement, alignment, and programming error, as well as defining more accurate measurement data.

Both Euler angle and quaternion attitude representation were used in the development of the CelNav algorithm. CelNav was first developed using Euler angle attitude representation due to the ease of understanding and without consideration for computational efficiency. The stage of development was run with all sensor and alignment data given in quaternions to decrease computational effort. This stage of development tested the speed of the CelNav algorithm using lower memory requirement and the possibility of less access to the Central Processing Unit (CPU). Both methods performed very well with little difference in simulation run time. However, this may not be true when run on a slower space based system.

2.3 Model Development and Methods for Extraction of Navigational Coordinates

This section describes how sensor models are developed as well as different extraction methods, for latitude, longitude, heading, tilt and slope. There are two sensor models and three possible extraction methods that can be used. The sensor models represent the accelerometer, denoted as Γ, and star tracker, denoted as Δ. The extracted navigational coordinates are latitude(λ), longitude(ϕ) and heading(ϵ).

The specific extraction methods used are dependent upon what sensor measurements or derived information are available. This methodology is summarized as:

1. Given Γ and Δ: Extract $\alpha, \beta, \epsilon, \lambda,$ and ϕ
2. Given α, β and Δ: Extract $\lambda, \phi,$ and ϵ
3. Given $\lambda, \phi,$ and Δ: Extract $\alpha, \beta,$ and ϵ

Where α denotes tilt and β denotes slope.

The following notation is used throughout this thesis:

$$[M_A]^A_{Body} \rightarrow \text{Accelerometer Alignment with Respect to Body Frame}$$

$[M_A]^A_{Body}$ is the transformation matrix relating coordinate frame of the accelerometer (A) with respect to coordinate frame Body (vehicle/astronaut) (Body). That is:

$$[M_A]^A_{Body} \rightarrow \text{Accelerometer Alignment with Respect to Body Frame}$$

Likewise,

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix}_{Body} = \begin{bmatrix} M_A \end{bmatrix}^A_{Body} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}_A$$

$[M_A]^C_{Body}$ is the transformation matrix relating coordinate frame of the star tracker (A) with respect to coordinate frame (Body). That is:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix}_{Body} = \begin{bmatrix} M_A \end{bmatrix}^C_{Body} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}_C$$

The planetary body's gravity model can be approximated by $[\Omega]^{SD}_{Inertial}$. A gravity model is a map that shows the varying gravity fields of a planetary body that can be caused by areas of greater mass concentration. This can be shown as in the case of craters, due to the greater gravity at the base of a crater than along its ridge.[10] Another effect that can be shown is that there is less gravity around the equatorial region. For this research $[\Omega]^{SD}_{Inertial}$ will be approximated at identity. It is first necessary to show the transformation to $[\Omega]^{SD}_{Inertial}$.

$$\begin{aligned} [\Omega]_{SD,I} = & \; [\Omega]_{MCMF,I}[\Omega]_{SC,MCMF}[\Omega]_{SD,SC} \\ = & \; \mathrm{I}^3 \end{aligned} \tag{2.1}$$

Where, Moon Centered Moon Fixed (MCMF), is the coordinate from fixed to the center of the Moon and rotates with it. Next Selenocentric (SC), is the coordinate from fixed to the center of the Moon but free to independently rotate. Finally, Selenodetic (SD) is the coordinate frame used on the surface of the Moon.

First it will be shown how each sensor model (accelerometer and star tracker) is developed then the necessary derived matrices will be derived and shown.

2.3.1 Accelerometer Model

The accelerometer model is based on the need for three 3-D accelerometer that outputs in the form:

$$\text{Acc}_{\text{measured}} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = xi + yj + zk \tag{2.2}$$

Each of the three axes of the rover's body (x, y, and z) are oriented with the accelerometer axes. The resulting models for the accelerometers oriented in the x, y, and z axes are described below:

Formulation of the x-oriented 3-D accelerometer (i.e. column 1)

Each column of the accelerometer transformation matrix $[M_A]^A_{Body}$ must be derived individually, such that, first it is necessary to describe the x-vector by calculating η_x, and θ_1. Where the rotation vector about the x is:

$$\eta_x^{'} = i \text{ x } (xi + yj + zk) = yk - zj \tag{2.3}$$

Next normalizing the $\eta_x^{'}$ produces:

$$\eta_x = \frac{yk - zj}{\sqrt{y^2 + z^2}} \tag{2.4}$$

$$\theta_1 = cos^{-1}[i \cdot \frac{xi + yj + zk}{\sqrt{x^2 + y^2 + z^2}}] = cos^{-1}\frac{x}{\sqrt{x^2 + y^2 + z^2}} \tag{2.5}$$

This can then be related to a quaternion such that:

$$q_1 = \begin{bmatrix} sin\frac{\theta_1}{2} * 0 \\ sin(\frac{\theta_1}{2}) * \frac{-z}{\sqrt{y^2+z^2}} \\ sin(\frac{\theta_1}{2}) * \frac{y}{\sqrt{y^2+z^2}} \\ cos(\frac{\theta_1}{2}) \end{bmatrix} \tag{2.6}$$

Finally using the conversion formula for quaternions to Euler angles found in Appendix F:

$$\Gamma_1 = \begin{bmatrix} x \\ \frac{-y}{\sqrt{y^2+z^2}} * \sqrt{1-x^2} \\ \frac{-z}{\sqrt{y^2+z^2}} * \sqrt{1-x^2} \end{bmatrix} \tag{2.7}$$

Formulation of the y-direction (i.e. column 2)

$$\eta_y^{'} = j \text{ x } (xi + yj + zk) = -xk + zi \tag{2.8}$$

Next normalizing the $\eta_y^{'}$ produces:

$$\eta_y = \frac{-xk + zi}{\sqrt{x^2 + z^2}} \tag{2.9}$$

$$\theta_1 = cos^{-1}[j \cdot \frac{xi + yj + zk}{\sqrt{x^2 + y^2 + z^2}}] = cos^{-1}\frac{y}{\sqrt{x^2 + y^2 + z^2}} \tag{2.10}$$

This can then be related to a quaternion such that:

$$q_2 = \begin{bmatrix} sin(\frac{\theta_1}{2}) * \frac{-x}{\sqrt{x^2+z^2}} \\ sin\frac{\theta_1}{2} * 0 \\ sin(\frac{\theta_1}{2}) * \frac{z}{\sqrt{x^2+z^2}} \\ cos(\frac{\theta_1}{2}) \end{bmatrix} \tag{2.11}$$

Again using the conversion formula for quaternions to Euler angles found in Appendix E:

$$\Gamma_2 = \begin{bmatrix} \frac{-x}{\sqrt{x^2+z^2}} * \sqrt{1-y^2} \\ y \\ \frac{-z}{\sqrt{x^2+z^2}} * \sqrt{1-y^2} \end{bmatrix} \tag{2.12}$$

Formulation of the z-direction (i.e. column 3)

$$\eta_z^{'} = k \text{ x } (xi + yj + zk) = xj - yi \tag{2.13}$$

Next normalizing the η_z' produces:

$$\eta_z = \frac{xj - yi}{\sqrt{x^2 + y^2}} \tag{2.14}$$

$$\theta_1 = cos^{-1}[k \cdot \frac{xi + yj + zk}{\sqrt{x^2 + y^2 + z^2}}] = cos^{-1}\frac{z}{\sqrt{x^2 + y^2 + z^2}} \tag{2.15}$$

This can then be related to a quaternion such that:

$$q_3 = \begin{bmatrix} sin(\frac{\theta_1}{2}) * \frac{x}{\sqrt{x^2+y^2}} \\ sin(\frac{\theta_1}{2}) * \frac{-y}{\sqrt{x^2+y^2}} \\ sin\frac{\theta_1}{2} * 0 \\ cos(\frac{\theta_1}{2}) \end{bmatrix} \tag{2.16}$$

Lastly,

$$\Gamma_3 = \begin{bmatrix} \frac{-x}{\sqrt{x^2+y^2}} * \sqrt{1 - z^2} \\ \frac{-y}{\sqrt{x^2+y^2}} * \sqrt{1 - z^2} \\ z \end{bmatrix} \tag{2.17}$$

Total Γ Matrix This results in the following combined Γ Matrix:

$$\Gamma_{Total} = \begin{bmatrix} \Gamma1 & \Gamma2 & \Gamma3 \end{bmatrix} \tag{2.18}$$

$$\Gamma_{Total} = \begin{bmatrix} x & \frac{-x}{\sqrt{x^2+z^2}} * \sqrt{1 - y^2} & \frac{-x}{\sqrt{x^2+y^2}} * \sqrt{1 - z^2} \\ \frac{-y}{\sqrt{y^2+z^2}} * \sqrt{1 - x^2} & y & \frac{-y}{\sqrt{x^2+y^2}} * \sqrt{1 - z^2} \\ \frac{-z}{\sqrt{y^2+z^2}} * \sqrt{1 - x^2} & \frac{-z}{\sqrt{x^2+z^2}} * \sqrt{1 - y^2} & z \end{bmatrix} \tag{2.19}$$

Through the use of three 3-D accelerometers it is possible to reduce an infinite number of possible rotation matrices to two unique rotation matrices. The two possible matrices are the positive and negative of each other, which mean they both perform the rotation to get the correct location, just in opposite directions.

2.3.2 Star Tracker Model

As previously stated the star tracker quaternion is in the form:

$$\text{StarTracker}_{\text{measured}} = \begin{bmatrix} q1 \\ q2 \\ q3 \\ q0 \end{bmatrix} \tag{2.20}$$

where,

$$q = \begin{bmatrix} \sin(\frac{\theta}{2}) * \cos(\beta_x) \\ \sin(\frac{\theta}{2}) * cos(\beta_y) \\ \sin(\frac{\theta}{2}) * cos(\beta_z) \\ \cos(\frac{\theta}{2}) \end{bmatrix} \tag{2.21}$$

Again the quaternions are in the form of the vector first (q1,q2,q3) and then the scaler (q0),where θ is the rotation angle and $\cos(\beta_x), \cos(\beta_y), \text{and} \cos(\beta_z)$ are the direction cosines locating the axes of rotation.

Another way to understand the quaternions is to convert them into a direction cosine matrix or transformation matrix. This can be accomplished by first rewriting the unit quaternion as follows:

$$\begin{bmatrix} q1 \\ q2 \\ q3 \\ q0 \end{bmatrix} = \begin{bmatrix} Q(1) \\ Q(2) \\ Q(3) \\ Q(4) \end{bmatrix} \tag{2.22}$$

With this change the individual quaternion values can be assembled into: [32]

$$M = \begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} \tag{2.23}$$

where,

$$Q_1 = \begin{matrix} Q(1,1)^2 - Q(2,1)^2 - Q(3,1)^2 + Q(4,1)^2 \\ 2 * (Q(1,1) * Q(2,1) - Q(3,1) * Q(4,1)) \\ 2 * (Q(1,1) * Q(3,1) + Q(2,1) * Q(4,1)) \end{matrix} \tag{2.24}$$

$$Q_2 = \begin{array}{c} 2*(Q(1,1)*Q(2,1)+Q(3,1)*Q(4,1)) \\ -Q(1,1)^2+Q(2,1)^2-Q(3,1)^2+Q(4,1)^2 \\ 2*(Q(2,1)*Q(3,1)-Q(1,1)*Q(4,1)) \end{array} \tag{2.25}$$

$$Q_3 = \begin{array}{c} 2*(Q(1,1)*Q(3,1)-Q(2,1)*Q(4,1)) \\ 2*(Q(2,1)*Q(3,1)+Q(1,1)*Q(4,1)) \\ -Q(1,1)^2-Q(2,1)^2+Q(3,1)^2+Q(4,1)^2 \end{array} \tag{2.26}$$

2.3.3 Rotation Correction and Alignment Matrices

An a priori source needed is the "U" transformation matrix. This matrix converts from the standard East-North-Up (ENU) convention to the North-East-Down (NED) convention used in this paper. The matrix can be written as follows:

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} E \\ N \\ U \end{bmatrix} \Rightarrow \begin{bmatrix} N \\ E \\ D \end{bmatrix} = \begin{bmatrix} U \end{bmatrix} \begin{bmatrix} E \\ N \\ U \end{bmatrix} \tag{2.27}$$

Other matrices used are the body alignment matrices. These matrices represent the alignment of the sensors with respect to the vehicle body frame, designated as $[M]^A_{Body}$ and $[M]^C_{Body}$ for the accelerator and star tracker, respectively. For this research, and without loss of generality, the body alignment matrices for both the accelerometer and star tracker are assumed to be identity. This will not hold true for real systems, however, as the transportation and landing of a space vehicle will skew the alignment matrix from its initial identity configuration, resulting in alignment error.

2.3.4 Extraction of Navigational Coordinates

The extraction methods given navigational and orientation data are discussed in this section. That is,

1. Given Γ and Δ: Extract $\alpha, \beta, \epsilon, \lambda,$ and ϕ

2. Given α, β and Δ: Extract $\lambda, \phi,$ and ϵ

3. Given $\lambda, \phi,$ and Δ: Extract $\alpha, \beta,$ and ϵ

2.4 Extracting Local Coordinates from Accelerometer and Star Tracker Data

$[\mathrm{H}(\epsilon)]_{NED}^{Down}$ is defined as the heading transformation matrix relating "Down" to NED coordinate frame and can be calculated such that,

$$[H(\epsilon)]_{NED}^{Down} = \begin{bmatrix} \cos\epsilon & \sin\epsilon & 0 \\ -\sin\epsilon & \cos\epsilon & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.28}$$

$[\mathrm{LL}(\lambda, \phi)]_{SD}^{NED}$ is defined as the cosine latitude and longitude matrix describing the attitude change of NED coordinate frame with respect to the selenodetic coordinate frame:

$$LL(\lambda, \phi)]_{SD}^{NED} = \begin{bmatrix} \sin\lambda & \cos\lambda & 0 \\ -\cos\lambda\sin\phi & -\sin\lambda\cos\phi & \cos\phi \\ \cos\lambda\cos\phi & \sin\lambda\cos\phi & \sin\phi \end{bmatrix} \tag{2.29}$$

Combining these equations create the following relationship:

$$[H(\epsilon)]_{NED}^{Down}[LL(\lambda, \phi)]_{SD}^{NED} = \begin{bmatrix} \cos\epsilon & \sin\epsilon & 0 \\ -\sin\epsilon & \cos\epsilon & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sin\lambda & \cos\lambda & 0 \\ -\cos\lambda\sin\phi & -\sin\lambda\cos\phi & \cos\phi \\ \cos\lambda\cos\phi & \sin\lambda\cos\phi & \sin\phi \end{bmatrix} \tag{2.30}$$

Then, combining Equation 2.29 and 2.30:

$$\begin{bmatrix} -\cos\epsilon\sin\lambda - \sin\epsilon\cos\lambda\sin\phi & \cos\epsilon\cos\lambda - \sin\epsilon\sin\lambda\sin\phi & \sin\epsilon\cos\phi \\ -\sin\epsilon\sin\lambda + \sin\epsilon\cos\lambda\sin\phi & -\sin\epsilon\cos\lambda - \cos\epsilon\sin\lambda\sin\phi & \cos\epsilon\cos\phi \\ \cos\lambda\cos\phi & \sin\lambda\cos\phi & \sin\phi \end{bmatrix} = [\Phi_A]_{SD}^{Down} \tag{2.31}$$

This equation is equivalent to the output as it can be calculated from sensors and "a priori" alignment matrices such that:

$$[\Phi_A]_{SD}^{Down} = [\Gamma]_A^{Down}[M_A]_{Body}^A([M_C]_{Body}^C)^T([\Delta]_C^{Inertial})^T([\Omega]_{Inertial}^{SD})^T \tag{2.32}$$

In the above equation Γ and Δ represent the accelerometer and star tracker respectively. Also included are $[M_A]_{Body}^A$ and $[M_C]_{Body}^C$ which represent the accelerometer and star tracker alignments with respect to the body coordinate frame. The above matrix $[\Phi_A]_{SD}^{Down}$ is the matrix from which it is possible to calculate λ, ϕ, and ϵ. These can be extracted as follows:

$$\begin{aligned}
\lambda &= \tan^{-1}\left[\frac{\Phi_A(3,2)}{\Phi_A(3,1)}\right] \\
\phi &= \tan^{-1}\left[\frac{\Phi_A(3,3)}{\sqrt{\Phi(2,3)^2 + \Phi(1,3)^2}}\right] \\
\epsilon &= \tan^{-1}\left[\frac{\Phi_A(2,3)}{\Phi_A(1,3)}\right]
\end{aligned} \tag{2.33}$$

Note that it can be seen in Equation 2.34 that it would be also possible to extract ϕ as seen:

$$\phi = \sin^{-1}[\Phi(3,3)] \tag{2.34}$$

This would be a valid solution except that $\sin^{-1}()$ requires quadrant determination. This is due to having the same sign in both the first and second quadrant.

2.4.1 Extracting Tilt, Slope, and Star Tracker Given Longitude, Latitude and Heading

As stated above:

$$[\Phi_A]_{SD}^{Down} = [\Gamma]_A^{Down}[M_A]_{Body}^A([M_C]_{Body}^C)^T([\Delta]_C^{Inertial})^T([\Omega]_{Inertial}^{SD})^T \tag{2.35}$$

It is possible to further de-construct $[\Gamma]_A^{Down}[M_A]_{Body}^A$ and define it as:

$$[ST(\alpha,\beta)]_{Down}^{Body} = ([\Gamma]_A^{Down}[M_A]_{Body}^A)^T \tag{2.36}$$

Where,$[ST(\alpha,\beta)]_{Down}^{Body}$, equals:

$$[ST(\alpha,\beta)]_{Down}^{Body} = \begin{bmatrix} \cos(\beta) & \sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) \\ 0 & \cos(\alpha) & \sin(\beta) \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{2.37}$$

Such that $[\Phi_{\alpha\beta}]_{SD}^{Down}$ equals:

$$\begin{aligned} [\Phi_{\alpha\beta}]_{SD}^{Down} &= ([ST(\alpha,\beta)]_{Down}^{Body})^T([\Omega]_{Inertial}^{SD}[\Delta]_C^{Inertial}[M_C]_{Body}^C) \\ &= ([ST(\alpha,\beta)]_{Down}^{Body})^T([M_C]_{Body}^C)^T([\Delta]_C^{Inertial})^T([\Omega]_{Inertial}^{SD})^T \end{aligned}$$

and,

$$[\Phi_A]_{SD}^{Down} = [\Phi_{\alpha\beta}]_{SD}^{Down} \tag{2.38}$$

The rest follows the same as above,

$$\begin{bmatrix} -\cos\epsilon\sin\lambda - \sin\epsilon\cos\lambda\sin\phi & \cos\epsilon\cos\lambda - \sin\epsilon\sin\lambda\sin\phi & \sin\epsilon\cos\phi \\ -\sin\epsilon\sin\lambda + \sin\epsilon\cos\lambda\sin\phi & -\sin\epsilon\cos\lambda - \cos\epsilon\sin\lambda\sin\phi & \cos\epsilon\cos\phi \\ \cos\lambda\cos\phi & \sin\lambda\cos\phi & \sin\phi \end{bmatrix} = [\Phi_{\alpha\beta}]_{SD}^{Down} \tag{2.39}$$

Again to calculate λ, ϕ, and ϵ. They can be extracted as follows:

$$\begin{aligned}
\lambda &= \tan^{-1}\left[\frac{\Phi_A(3,2)}{\Phi_A(3,1)}\right] \\
\phi &= \tan^{-1}\left[\frac{\Phi_A(3,3)}{\sqrt{\Phi(2,3)^2+\Phi(1,3)^2}}\right] \\
\epsilon &= \tan^{-1}\left[\frac{\Phi_A(2,3)}{\Phi_A(1,3)}\right] \qquad (2.40)
\end{aligned}$$

2.4.2 Extracting Latitude, Longitude, and Star Tracker Given Tilt, Slope, and Heading

A third possibility is to extract tilt slope and heading from known latitude, longitude, and star tracker data.

$$([ST(\alpha,\beta)]_{Down}^{Body}H(\epsilon)]_{NED}^{Down})^T = ([U]_{ENU}^{NED})^T[LL(\lambda,\phi)]_{SD}^{ENU}[\Omega]_{Inertial}^{SD}[\Delta]_C^{Inertial}[M_C]_{Body}^C \quad (2.41)$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\lambda)\sin(\phi) & \sin(\lambda)\sin(\phi) & \cos(\phi) \\ \cos(\lambda)\cos(\phi) & \sin(\lambda)\cos(\phi) & \sin(\phi) \end{bmatrix} \Omega]_{Inertial}^{SD}[\Delta]_C^{Inertial}[M_C]_{Body}^C$$

$$= \begin{bmatrix} -\cos(\lambda)\sin(\phi) & -\sin(\lambda)\sin(\phi) & \cos(\phi) \\ -\sin(\phi) & \cos(\lambda) & 0 \\ -\cos(\lambda)\cos(\phi) & \sin(\lambda)\cos(\phi) & -\sin(\phi) \end{bmatrix} \Omega]_{Inertial}^{SD}[\Delta]_C^{Inertial}[M_C]_{Body}^C \quad (2.42)$$

It was previously stated that:

$$([ST(\alpha,\beta)]_{Down}^{Body}H(\epsilon)]_{NED}^{Down})^T =$$

$$\left(\begin{bmatrix} \cos(\beta) & \sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) \\ 0 & \cos(\alpha) & \sin(\beta) \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos\epsilon & \sin\epsilon & 0 \\ -\sin\epsilon & \cos\epsilon & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)^T \quad (2.43)$$

$$A_1 = \begin{pmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) \\ -\cos(\alpha)\sin(\epsilon) \\ \sin(\beta)\cos(\epsilon) + \cos(\beta)\sin(\alpha)\sin(\epsilon) \end{pmatrix} \quad (2.44)$$

$$A_2 = \begin{pmatrix} \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) \\ \cos(\alpha)\cos(\epsilon) \\ \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) \end{pmatrix} \tag{2.45}$$

$$A_3 = \begin{pmatrix} -\sin(\beta)\cos(\alpha) \\ -\sin(\alpha) \\ \cos(\beta)\cos(\alpha) \end{pmatrix} \tag{2.46}$$

$$= \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix}^T \tag{2.47}$$

Taking the transpose yields:

$$B_1 = \begin{pmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) \\ \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) \\ -\sin(\beta)\cos(\alpha) \end{pmatrix} \tag{2.48}$$

$$B_2 = \begin{pmatrix} -\cos(\alpha)\sin(\epsilon) \\ \cos(\alpha)\cos(\epsilon) \\ \sin(\alpha) \end{pmatrix} \tag{2.49}$$

$$B_3 = \begin{pmatrix} -\sin(\beta)\cos(\alpha) \\ \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) \\ \cos(\beta)\cos(\alpha) \end{pmatrix} \tag{2.50}$$

$$= \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix}^T \tag{2.51}$$

Thus, its is possible to extract tilt, slope, and heading as follows:

$$\alpha = \sin^{-1}[\Psi(3,2)]$$

$$\beta = \tan^{-1}\left[\frac{-\Psi_(3,1)}{\Psi(3,3)}\right]$$

$$\epsilon = \tan^{-1}\left[\frac{-\Psi_(1,2)}{\Psi(2,2)}\right] \tag{2.52}$$

2.4.3 Extracting Tilt and Slope

Here it will be shown how the tilt and slope matrices are assembled, and how information may be extracted from them. The tilt/slope matrix takes on the form:

$$[ST(\alpha,\beta)]^{Body}_{Down} = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\beta) & \sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) \\ 0 & \cos(\alpha) & \sin(\alpha) \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{2.53}$$

where,

$$[ST(\alpha,\beta)]^{Body}_{Down} = ([\Gamma^{Down}_{Acc}][M_A]^{Acc}_{Body})^T \tag{2.54}$$

such that

$$[\Gamma, M_A]^{Down}_{Body} = [\Gamma^{Down}_{Acc}][M_A]^{Acc}_{Body} \tag{2.55}$$

By examining Equation 2.55, it is possible to directly extract α and β:

$$\alpha = \tan^{-1}\left[\frac{[\Gamma, M_A](3,2)}{[\Gamma, M_A](2,2)}\right]$$

$$\beta = \tan^{-1}\left[\frac{-[\Gamma, M_A](1,3)}{[\Gamma, M_A](1,1)}\right]$$

Chapter III

Errors and Disturbances

CelNav was developed as an inexpensive and effective mobility solution for the navigation of extra planetary bodies. Lumped error analysis was shown by Thein et al.[2] to achieve a 50m accuracy with a maximum lumped uncertainty/sensor error of 5.93 arcsec. Current Lunar navigational requirements allow for navigational errors of 2000m.

CelNav has the capacity work as two different systems, a mobility solution and a fault detection algorithm. CelNav has the capability to check for correct heading, latitude, and longitude using different sensor data depending on what is available. Therefore, when a rover/astronaut can communicate with a ground station or an orbiting satellite, CelNav can be used as a local confirmation of latitude and longitude. Or, if a rover/astronaut suddenly loses contact with said satellite or ground station, current latitude and longitude will still be available for navigation due to CelNav.

By using the appropriate matrices it is possible to extract all necessary navigational data (latitude, longitude, and heading as well as tilt and slope) using

CelNav and accurate Γ and Δ matrices.

This chapter focuses on the expected errors associated with the CelNav navigational system, from an experimentally constructed star tracker system, Contained Attitude Star Tracking Sensor (CASTS), and a 3-D accelerometer, as well as errors associated with necessary initial conditions.

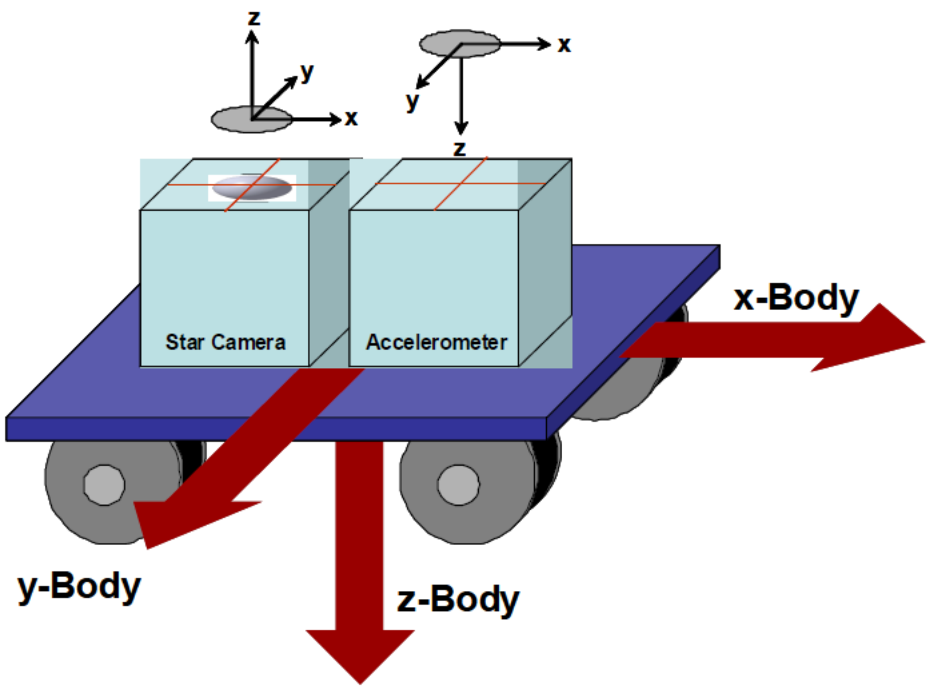

Figure 3.1: Rover Example [22]

In practical applications, a rover/astronaut would have an accelerometer mounted such that it would always point towards the center-of-gravity (COG) of the Moon and the star camera that points angled to the front and to the rear. This reasoning will be explained later in this chapter.

3.1 Accelerometer Errors

This section focuses on both measurement and alignment errors found in the accelerometer implementation.

An accelerometer triad is used to determine the tilt and slope of the astronaut or rover. In order to accurately determine both tilt and slope only one (1) 3-D accelerometer is needed to generate a unique matrix denoted by Φ as shown in Equation 2.35. From this unique Φ, latitude and longitude is extracted using a Γ matrix of the form of Equation 2.19. The expected total error of the 3-D accelerometers is expected to be on the order of 10^{-3}, as stated by Quinn [22] and Thein et al.[2].

The mounting of the accelerometer triad introduces a significant portion of the total error that is expected in the system. CelNav relies on accurate sensor placement to calculate the many required attitude transformations. In other words, proper sensor aliment is critical. For accurate navigation, the accelerometer alignment error is assumed orthogonal and is required to be within 60 arcsec. This includes the internal orientation of the 3-D accelerometer, as well as the mounting of the hardware to a rover or astronaut. This alignment error accounts for the greatest source of navigational error for the CelNav algorithm. Reducing this alignment error to even a conservative estimate of 10-20 arcsec, one may reduce the maximum navigation error in CelNav within 200-300 meters as can be seen in Figure 3.2. A rover or astronaut has a line of sight of 2000m[22]. In Figure 3.2 the three circles represent 1σ, 2σ, and 3σ, to be discussed in greater detain in Chapter 4. It can be seen that a great deal of the possible navigational errors fall within the 1σ circle, where 68.28% confidence interval, as calculated by CelNav.

With the greatest error being about 600m from the actual location. Through greater alignment error reduction or improved calibration navigational error can be greatly reduced.

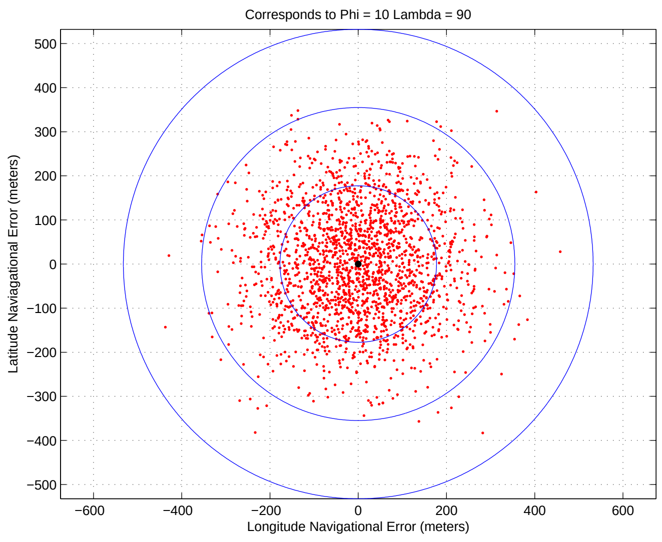

Figure 3.2: 10-20 Arcsec Accelerometer Alignment Noise

3.2 Simulated Star Tracker and Contained Attitude Star Tracking Sensor

Contained Attitude Star Tracking Sensor (CASTS) was developed by Tyler Wills [26] for use in swarm optimization. Further design and construction discussion can be found in Appendix C. CASTS was designed to be easily transportable for installation on multiple mobile platforms. CASTS contains camera sensor(s), an accelerometer and a "star field", canvas enclosure containing the colored "stars", to be discussed later. CASTS is attached to a mock Lunar rover, to be discussed later, containing a drive mechanism, camera platform and "star field" mounting.

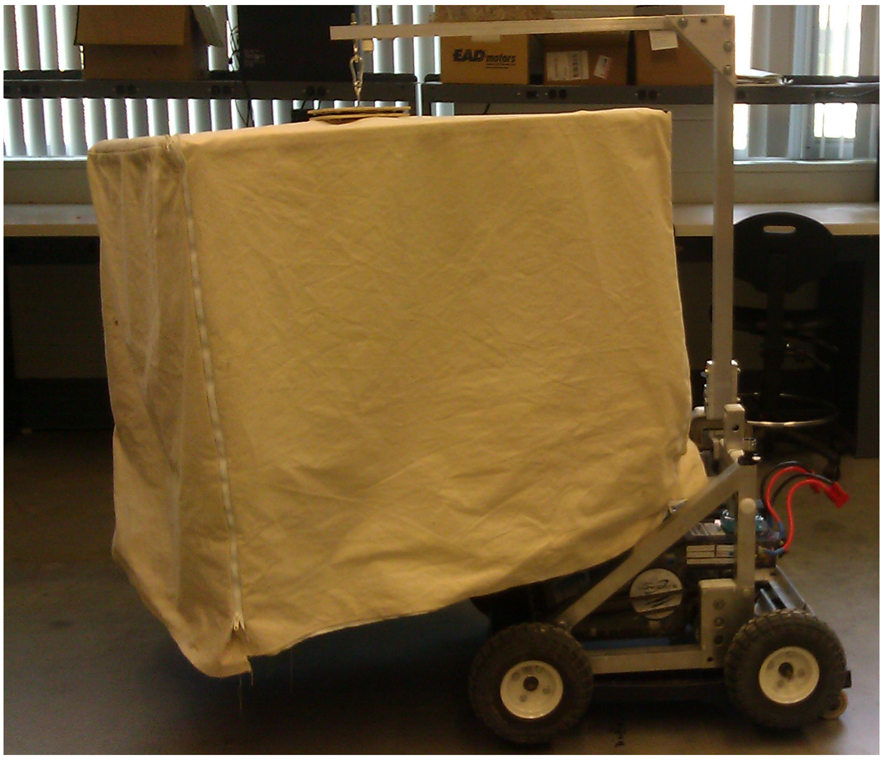

Figure 3.3: CASTS

CASTS, as described above, is a two part system: cameras/3-D accelerometer and a"star field". The initial difficulty is developing a compact sensor platform that could be easily moved. This required developing a hard mount for the cameras and 3-D accelerometer. This mounting allows for the alignment of the upward looking camera to the middle of the "star field", as seen in Figure 3.4, reasoning to be discussed later. Also important is the mounting of the accelerometer. The mounting platform need to be parallel with the body of the mobile platform, so that "down" can be determined with respect to the mobile platform's body. Proper mounting is critical to reducing the sensor alignment error, to be discussed later in this chapter.

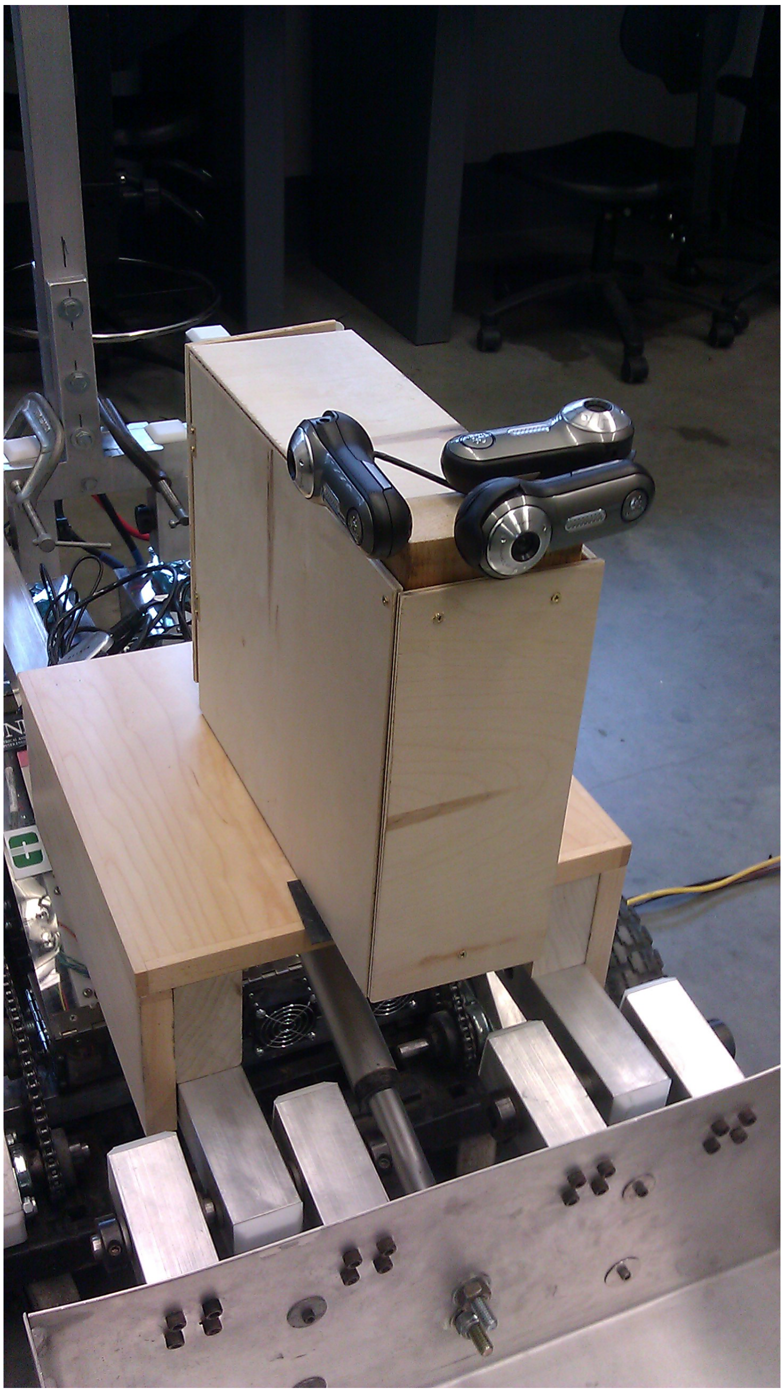

Figure 3.4: CASTS Camera Configuration

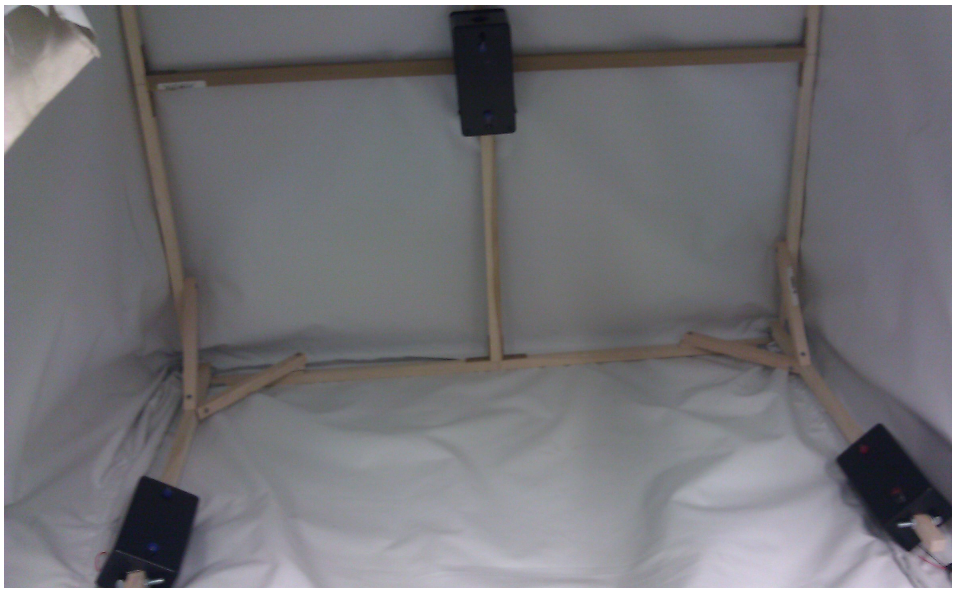

Figure 3.5: CASTS “Star Field” Enclosure

3.2.1 Measurement Error

In this section two different types of measurement error are discussed. First is the star tracker error that is artificially introduced in an analytical simulation. Next is the error found in the CASTS's sensor system implemented in the experimental test results. In practical applications two star cameras are necessary, with one angled towards the front of rover/astronaut and one angled to the rear. This is necessary to account for instances when one camera may have its field of view blocked, for example, by the local terrain.

The simulated star tracker quaternion is corrupted with a total of 60 arcsec of error; the error is distributed with 10 arcsec along the x-axis, 10 arcsec along the y, and 40 arcsec along the z, as per specification of NASA aerospace engineer David Quinn.[22] It is found that only increasing or decreasing the level of noise along the z-axis has any effect on the system; changes along and x and y-axes have no significant effect. Though this effect is almost unnoticeable, it does, in fact, add non-negligible error to the total navigational error. This can be seen in Table 3.1 with three distinct test cases to display the effects for 60 arcsec of measurement error dispersed over the x, y, and z axes.

Star Tracker Alignment Errors			
	Case 1	Case 2	Case 3
x-axis	40	10	10
y-axis	10	40	10
z-axis	10	10	40

Table 3.1: Latitude 89.9 and Longitude 90 - Test Case Criterion

In Figures 3.6, 3.7, and 3.8 it can be difficult to see a difference between the three figures. But, if the 3σ circles are examined it can be seen that the radius is larger for Figure 3.8 when compared to Figure 3.6 or 3.7. This difference may seem small but can matter greatly as more error conditions are added to the sensor models.

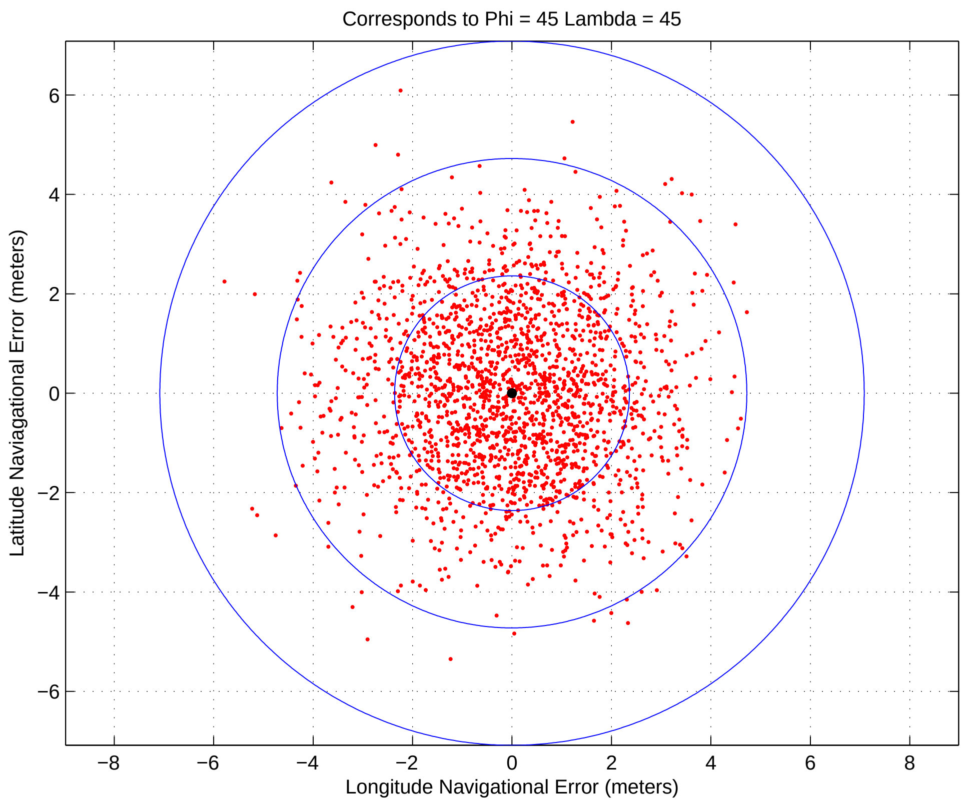

Figure 3.6: Case 1 - x-axis 40, y-axis 10, and z-axis 10

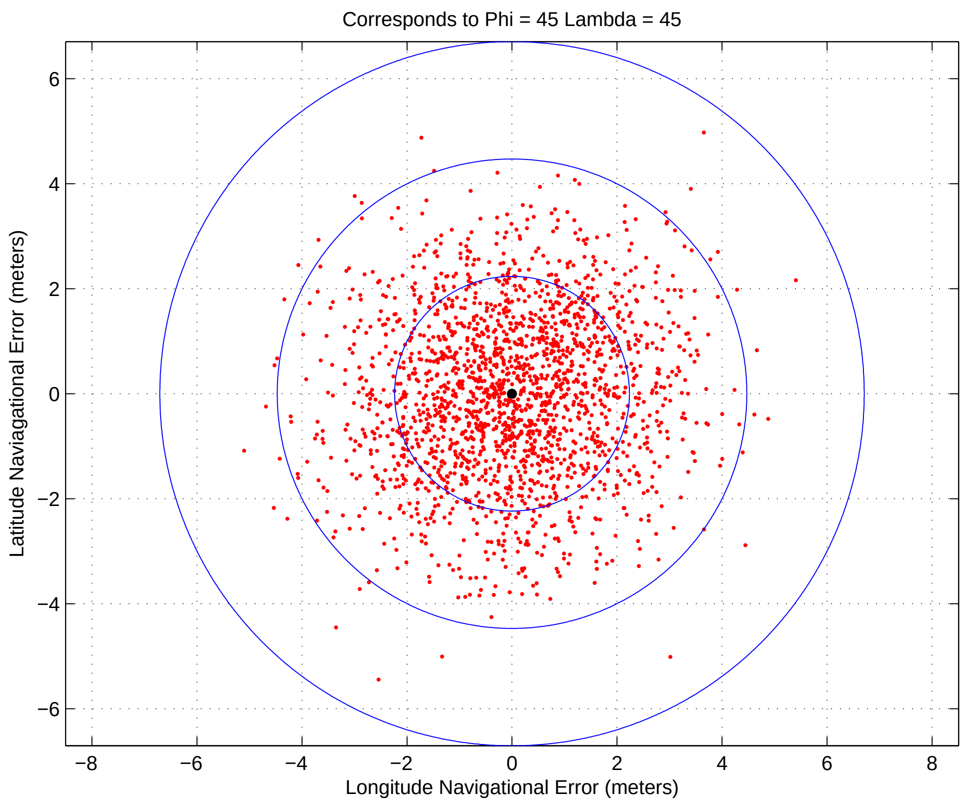

Figure 3.7: Case 2 - x-axis 10, y-axis 40, and z-axis 10

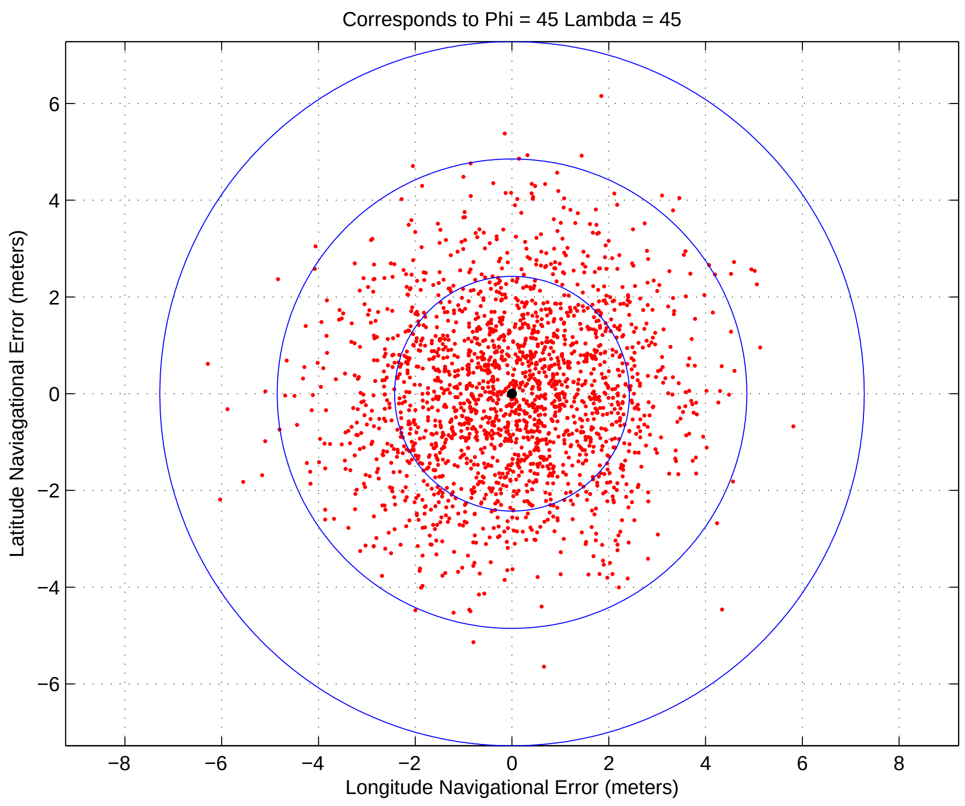

Figure 3.8: Case 3 - x-axis 10, y-axis 10, and z-axis 40

CASTS measurement error is highly dependent on proper calibration. This is due to colors being repeated around the outside and the"Star" patterns remaining constant. The colors of the "star field" are red, green and blue, with two blues being repeated on adjacent walls. An additional blue "star pattern" is mounted on the "roof" of the "star field" to represent heading. The "star field" pattern can be seen in Figure 3.9.

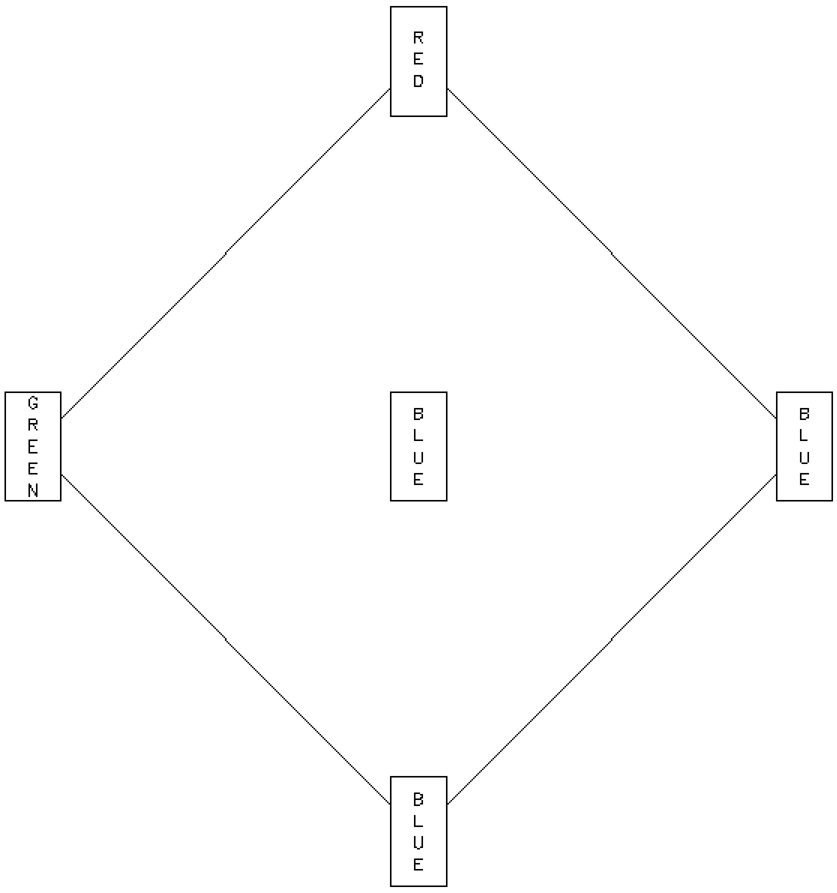

Figure 3.9: CASTS Experimental Platform

Another difficulty to note in CASTS is ensuring that the colored stars, (red, blue or green) are of the appropriate brightness and not contaminated by ambient light and/or are not overly bright. Additional error is due the cameras' not being able to accurately determine the exact center of each "star" due to the pixelation of the image. This can be accounted for by dimming the LEDs' intensity and making the color deeper. It may be possible by using different image processing techniques to reduce this measurement error and significantly increase accuracy.

Again it is important to note that if the calibration is performed without any tilt in the "star field," measurements will be erratic. But if the calibration is done with a moderate tilt in the "star field", samples are less erratic. For example if the first result could be 70deg north latitude the second could be 45$\circ$ north latitude with level calibration, but with the moderate tilt the first result could be 70deg north latitude the second could be 70.5$\circ$ north latitude without moving the CASTS experimental setup. This is due to the method used in the digital image processing of CASTS. When the "star field" is lined up in a linear fashion, as seen in Figure 3.10(A), during calibration, CASTS has a difficulty determining accurate latitude and longitude. This is most likely due to an inability to accurately determine distance between the "stars" in the "star field" when linearly aligned. But calibrating CASTS with the "star field" at an angle determining distances between "stars" is easier, thus minimizing erratic samples. To further reduce measurement inconsistencies in the environment such as camera blind spots, three stereoscopic cameras, are used. Thus by seeing two walls or a wall and the ceiling it is possible to get a unique measurement.

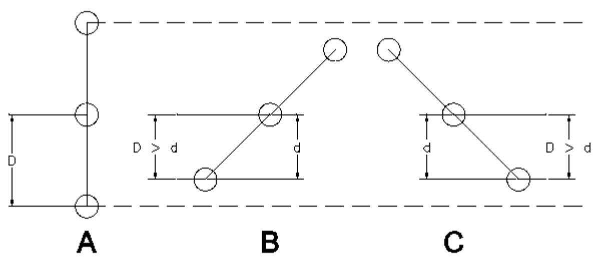

Figure 3.10: CASTS Experimental Star Alignment

A final difficultly is keeping the "star field" facing the same direction while rotating only the attached rover. Rotating the "star field" is similar to having incorrect star tracking maps, such that and data derived/calculated with the rotated field cannot be trusted. This difficulty produces a similar effect to the previous difficulty with improper calibration.

3.2.2 "Star Tracker" Alignment Error

In this section two different types of alignment error are discussed: (1) star tracker alignment error used in the analytical simulation, and (2) alignment error.

For simulation purposes the star tracker alignment matrix is constructed to have a max of 60 arcsec of artificially corrupted error. It is found that increasing and decreasing the amount of noise in this measurement has negligible effect on the total navigational error. Again, although this effect is almost unnoticeable, it does, in fact, add non-negligible error to the total navigational error, not shown here.

CASTS alignment error is highly dependent on the rigidity of the experimental setup.

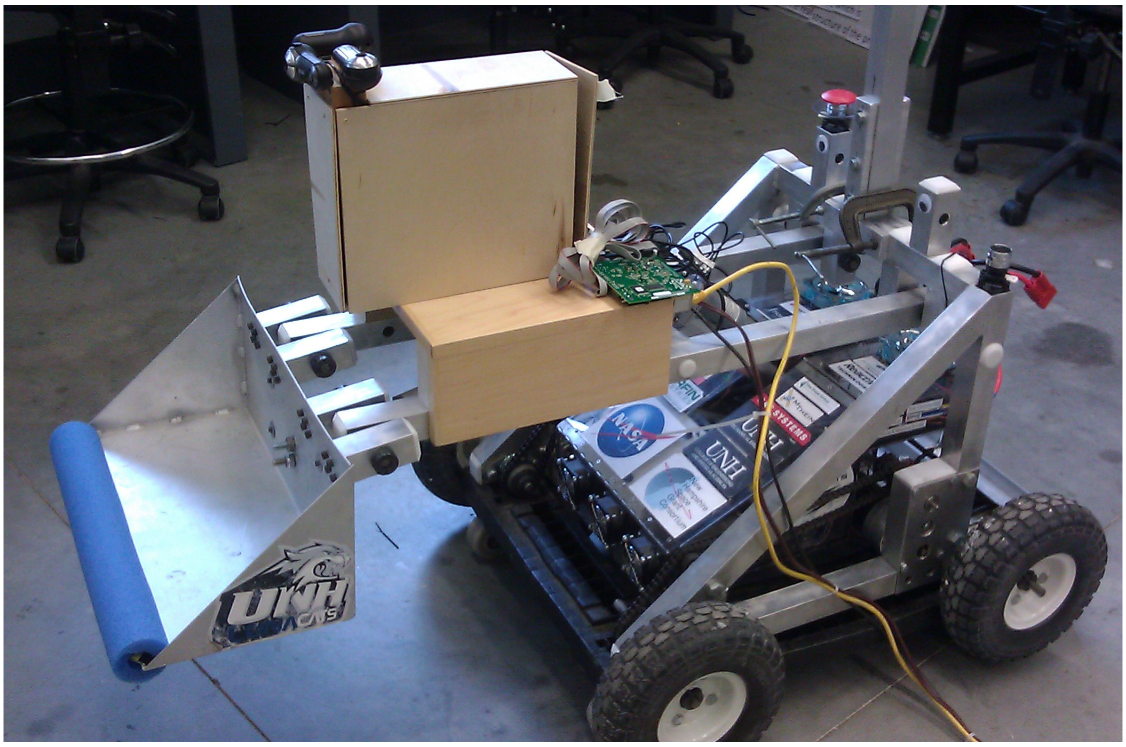

Figure 3.11: CASTS Experimental Star Setup

As can be seen in Figure 3.11 placement of the cameras inside the CASTS platform must be consistent. If the CASTS "star cameras" are not placed properly it is possible that some of the "star field" may not be within the viewing angle of the "star cameras" as seen in Figure 3.12. Each of three cameras face forward, right(not shown in Figure 3.12) and up. If any of these cameras move during testing the calibration is invalid, and needs to be repeated.

Again, placing the "star cameras" consistently in the same location will produce repeatable results.

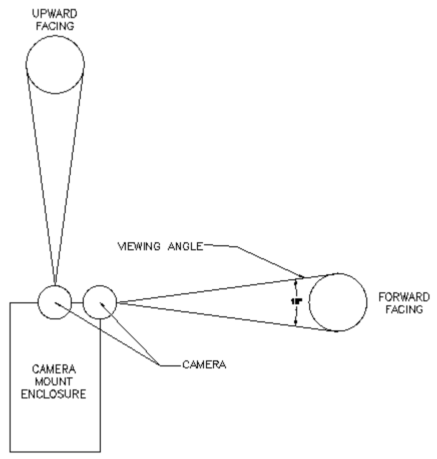

Figure 3.12: CASTS Star Camera Viewing Angle

3.3 Initial Conditions

An important assumption that is made in the implementation of CelNav algorithm into a dynamic navigation system is that the last valid location, described with latitude and longitude coordinates, are known. This is important due to implementation of dynamic observers to improve the accuracy of CelNav results. Observers are being implemented to help "clean up" the data generated by CelNav, for use in feedback for on-board controllers. These observers will be discussed at length in Chapter 7. For analytical simulation and experimental testing purposes, a priori data for latitude and longitude can be extracted from an Inertial Navigation System (INS), GPS, or other methods for determining location. CelNav, itself, does not require accurate initial conditions. However, the more accurate the initial conditions, the faster the implemented observers are be able to converge, returning accurate location data.

3.4 Inertial Navigation System (INS)

During testing it is found that the CASTS "star tracker" is impractical as it does not yield consistent "measurements." Due to the inconsistency in the ability to calibrate CASTS it is determined that an Inertial Measurement Unit (IMU) containing a GPS, 3-D magnetometer, and 3-D accelerometer should be used in place of a CASTS experimental setup to mimic the sensor output from a star tracker. Additional information can be found in Appendix C.

Chapter IV

CelNav Performance Analysis

4.1 Monte Carlo Analysis

Monte Carlo is the name associated with a mathematical technique developed by scientists working at Los Alamos National Lab in 1940.[3] The solutions sought through Monte Carlo simulations form a statistical answer; these are governed by the laws of chance. A good use of Monte Carlo analysis is when the answer is known a priori and it is necessary to find out how accurately an experiment is to providing said answer. Monte Carlo Analysis develops a range of possible answers which increases in accuracy with the increase of the number of experiments performed. According to Kalos, Monte Carlo can be defined as "[a] method that involves deliberate use of random numbers in a calculation that has the structures of a stochastic process"; with stochastic process defined as "a sequence of states whose evolutions [are] determined by random events." [3] The Monte Carlo method, as a result, is an appropriate application for CelNav. In this chapter the simulation parameters of the analytic simulations, as well as random noise levels, are discussed.

This testing "field" studies a "global map" of the Lunar surface, specifically looking at latitudes and longitudes with great distances between, then moving to latitudes and longitudes approaching the polar regions where latitudes and longitudes converge. This chapter analyzes the navigational error associated with both expected and "worst case" noise levels and how Monte Carlo parameters are calculated.

4.2 Monte Carlo Parameters

For this set of Monte Carlo simulations three parameters are chosen: required sample size, incremental propagation and noise levels. The calculation of a confidence interval is not required because the required accuracy of the simulations is known.

4.2.1 Sample Size

The number of required sample points is dependent upon the predicted sampling error and some bounding term B such that [2]:

$$\varepsilon \approx B = 2\sqrt{0.25/n} = 1/\sqrt{n} \tag{4.1}$$

This can, in turn, be simplified to:

$$1/\varepsilon^2 \approx 1/B^2 = n. \tag{4.2}$$

Since between 2.5% and 3% error is expected, Equation 4.2 shows that approximately 1000 and 1600 samples are required for a large enough random sample pool. Table 4.1 shows that CelNav is able to produce 2028 samples in a single simulation of a chosen latitude and longitude. This greatly exceeds the calculated number of necessary samples 1000-1600, from the above equation.

	Number of Data Points per ϕ and λ
Minimum	1000
Simulated	2028

Table 4.1: Number of Simulated Data Points

In the future it may be necessary to take a larger sampling size with the inclusion of real sensors instead of simulated ones. This larger sample size is necessary to reduce the variance generated by sensors producing something other than what can be modeled as white noise.

4.2.2 Sample Locations

One must also choose which latitudes and longitudes to sample. It is important to note that for this Monte Carlo analysis, testing encompasses the entire Lunar surface in latitude and longitude. This is important for repeatability of certain samples, as the simulation should, for example, have the same sample distribution for a constant latitude while rotating the body about different longitudes. This should hold true for regions regardless of the distances between latitudes and longitudes, which can be found in Table 4.2:

Increments	Range
$5°$	$-80° \Rightarrow 80°$
$1°$	$80 \Rightarrow 89°$ and -80° $\Rightarrow -89°$
$.01°$	$89 \Rightarrow 90°$ and $-89° \Rightarrow -90°$

Table 4.2: Sampling Intervals

4.3 Uncertainties and Biases

Three noise cases are examined: no noise, expected noise, “worst case” noise. These levels are chosen due to the unknown levels of both sensor and alignment noise found in the potential mobility solution. Table 4.3 contains the maximum levels of expected noise in real world mobility solution:

Noise Type	**Sensor**	**Noise Magnitude**
Measurement	Accelerometer	$1 * e^{-6}$

Measurement	Star tracker	X=10 Arcsec
		Y=10 Arcsec
		Z=40 Arcsec

Alignment	Star Tracker	60 Arcsec
Alignment	Accelerometer	60 Arcsec

Table 4.3: Monte Carlo Noise Power

4.4 Data Distribution

Below is the standard form for a probability density function (PDF)[3]:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.3}$$

The mean(μ) and the standard deviation(σ) for a generalized data set for a latitude can be seen in Table 4.4: This mean and standard deviation for a generalized data

Statistics Example	
Mean(μ)	57.2001
Standard Deviation(σ)	508.1359

Table 4.4: Monte Carlo mean (μ) and standard deviation (σ)

set in Table 4.4 can be shown graphically in Figure 4.1. Figure 4.1 is the probability distribution function for the generalized latitude. This figure was generated by performing a simulation and plotting the navigational error for said latitude. This concludes that the noise as modeled in the CelNav algorithm presents itself as a gaussian distribution. Showing the above is necessary for the development of the

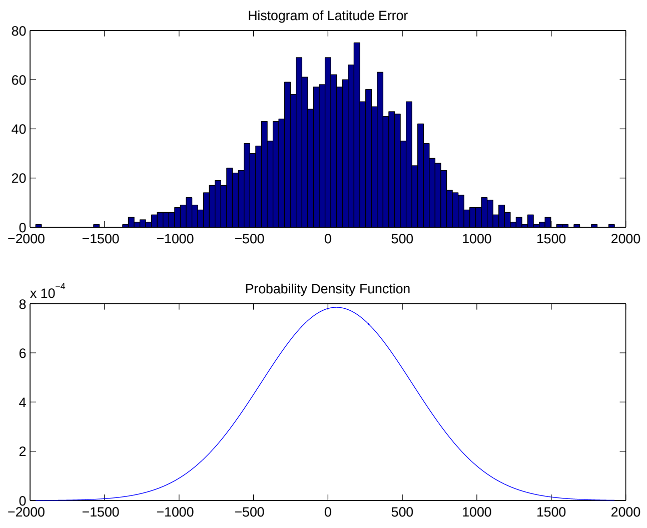

Figure 4.1: Probability Distribution Function - CelNav Simulation

Extended Kalman Filter (EKF). This is due to how the EFK models noise, which requires noise to be in a gaussian distribution.

4.5 Results

In this section it will be discussed how the three different noise levels work to decrease the accuracy of the CelNav algorithm. The noise levels are: no noise (zero (0) measurement and alignment noise), expected noise ($1*10^{-6}$ measurement and little to no alignment noise due to expected calibration), and maximum allowable noise, ($1*10^{-1}$ of measurement noise and 60 arcsec of alignment noise).

Three different Lunar regions will be explored in this thesis, including the equatorial and the polar regions (North Pole/South Pole). In each of the figures in this chapter, three circles are drawn which coincide with specific confidence intervals(1σ, 2σ, and 3σ) specified in Table 4.5. The latitude navigational error

Confidence Interval	
1-Sigma	68.26% Confidence Interval
2-Sigma	95.45% Confidence Interval
3-Sigma	99.73% Confidence Interval

Table 4.5: Confidence Intervals

can be calculated as:

$$\phi_{naverr} = \frac{\phi_e}{180} * R_{Moon} * \pi \tag{4.4}$$

The longitude navigational error can be calculated as:

$$\lambda_{naverr} = \frac{\lambda_e}{360} * (R_{moon} * cos(\phi_{true}) * \frac{\pi}{180}) * (2\pi) \tag{4.5}$$

Table 4.6 shows the criterion used in each test case.

	Location		Sensors Errors		
	Latitude	Longitude	Star Tracker Alignment	Accelerometer Alignment	Accelerometer Noise
Case 1	0	90	0	0	0
Case 2	0	90	0	0	$1 * e^{-6}$
Case 3	0	90	60 arcsec	60 arcsec	$1 * e^{-6}$

Table 4.6: Latitude 0 and Longitude 90 - Test Case Criterion

4.5.1 Test Case - Latitude 0 and Longitude 90

First the "no noise" simulation is examined. This simulation contains no measurement or alignment error. It is found that with this "no noise" case all the points fall to one point with no variability (less numerical error). As can be seen in Figure 4.2 below, with no error, neither measurement nor alignment errors, the location error is negligible (magnitude on the scale of 10^{-10}).

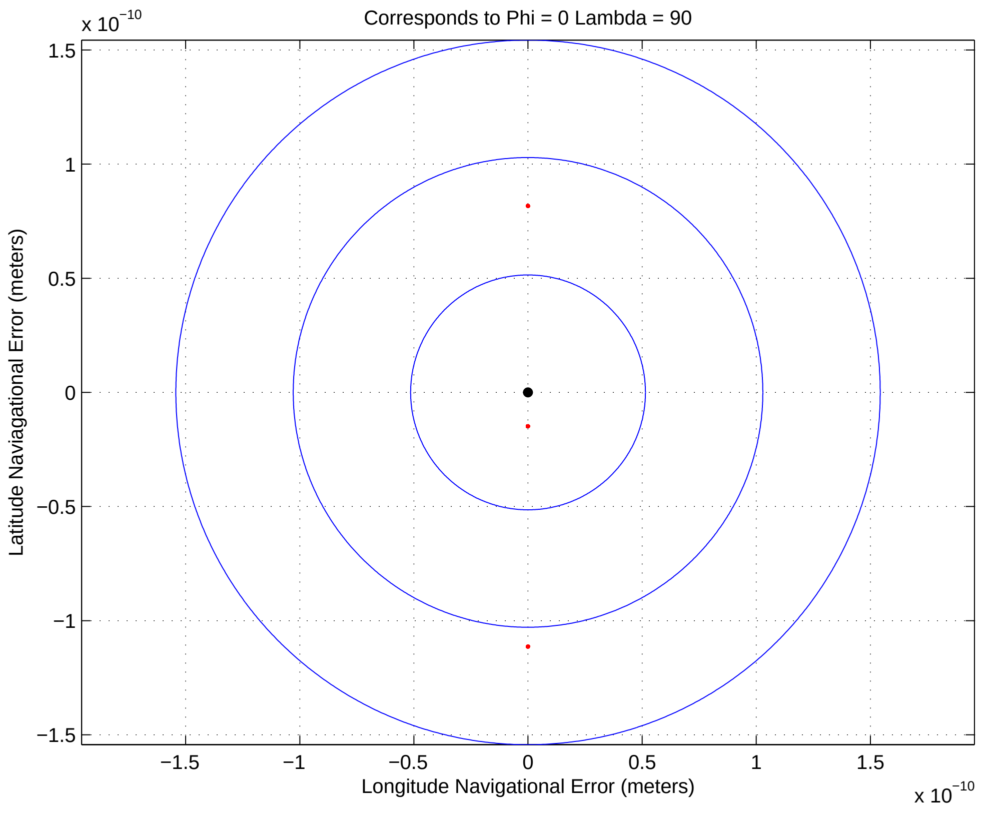

Figure 4.2: Case 1 (No Noise) - Latitude 0° Longitude 90°

Next, only accelerometer measurement noise for the “expected noise” simulation is discussed. It can be seen in Figure 4.3 that there is more scattering of locations as compared to Figure 4.2. With only marginal measurement noise almost all of the sample data points fall within a 4m radius 2-σ circle.

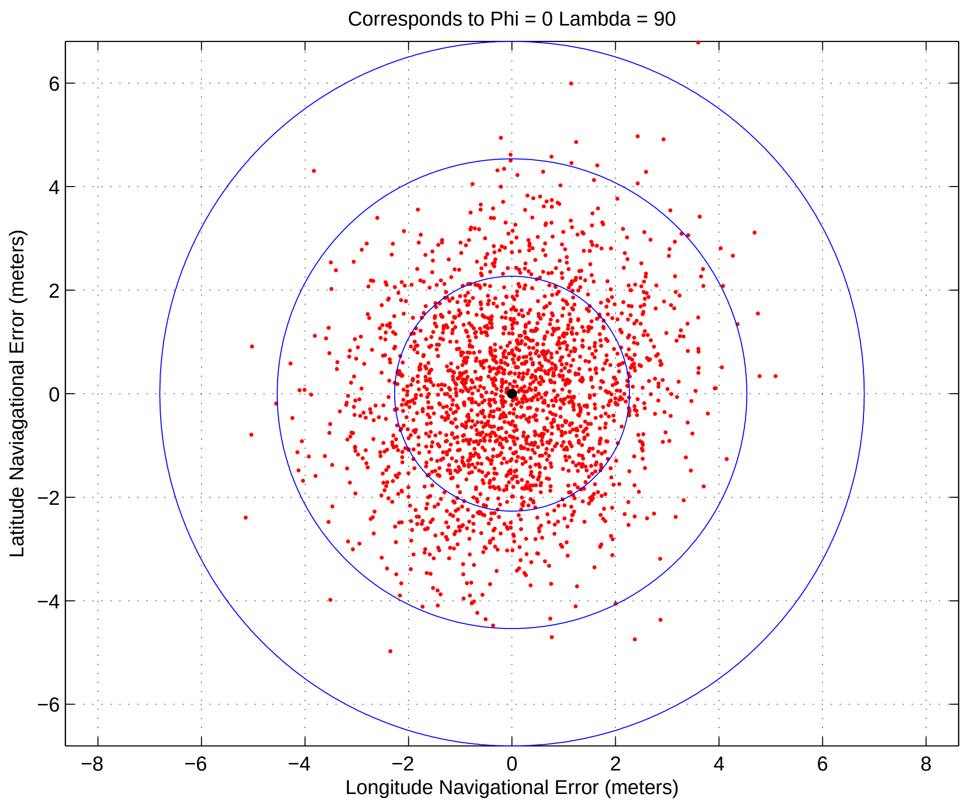

Figure 4.3: Case 2 (Accelerometer Measurement Noise) - Latitude 0° Longitude 90°

Figure 4.4 shows the "worst case" noise. Here, the navigational error increases to 2000m. This is a great increase in distance. However, an astronaut is able to clearly see upwards of 2500m on the Lunar surface.[22] As long as a majority of the possible locations sampled appear within the 3-σ circle, the error though large, is still deemed as acceptable.[22]

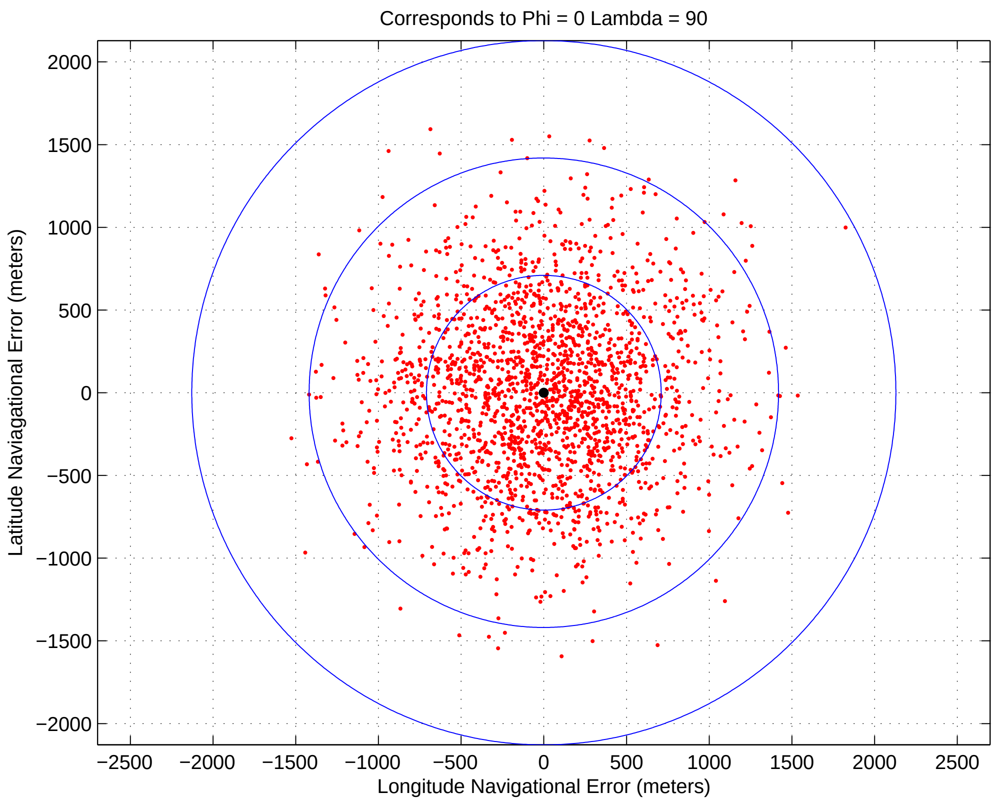

Figure 4.4: Case 3 (Worst Case Noise) - 0° Longitude 90°

4.5.2 Test Case - Latitude 89.9 Longitude 90

The next test case is chosen to be in close proximity to the Lunar north pole. The next set of test cases are shown in Table 4.7:

	Location		Sensors Errors		
	Latitude	Longitude	Star Tracker Alignment	Accelerometer Alignment	Accelerometer Noise
Case 1	89.9	90	0	0	0
Case 2	89.9	90	0	0	$1 * e^{-6}$
Case 3	89.9	90	60 arcsec	60 arcsec	$1 * e^{-6}$

Table 4.7: Latitude 89.9 and Longitude 90 - Test Case Criterion

Here it can again be seen in Figure 4.5 that with no error, neither measurement nor alignment error, navigational error decreased to a magnitude on the scale of 10^{-10}. This is the same magnitude that was seen previously in Figure 4.2, meaning that when instrumentation is even extremely close to the polar regions it is still possible to obtain acceptable navigational results, with reasoning to be explained later in this chapter.

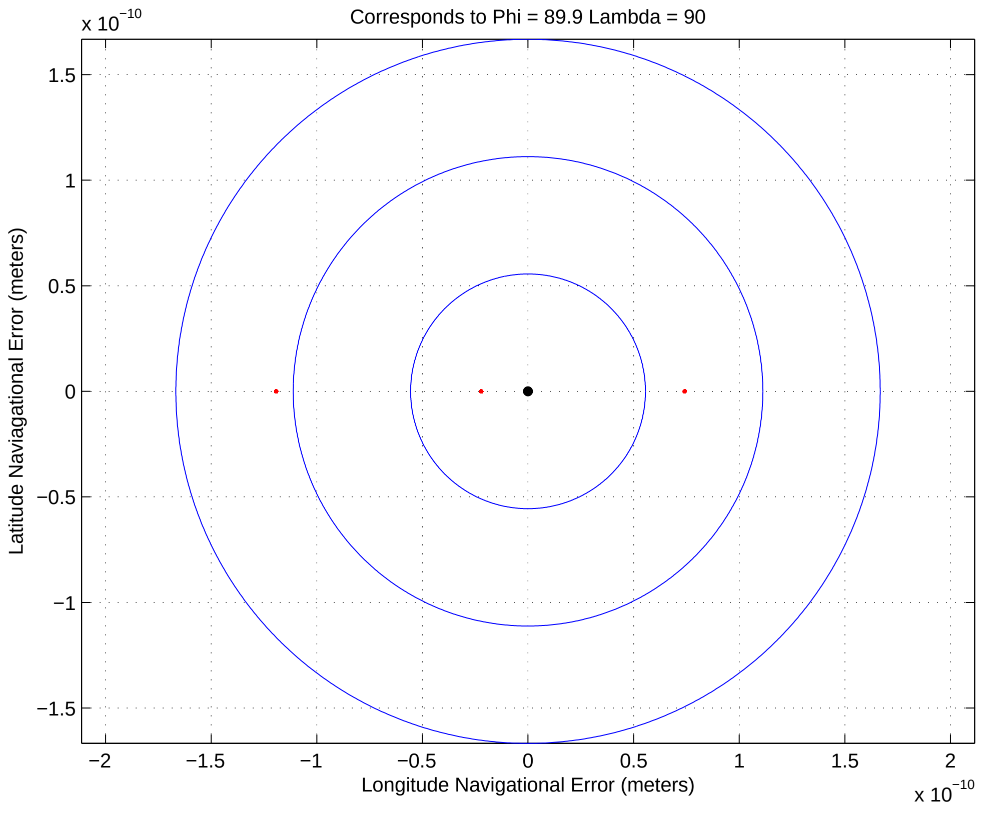

Figure 4.5: Case 1 (No Noise) - Latitude 89.9° Longitude 90°

Figure 4.6 shows an interesting occurrence at extreme latitudes such as that shown in Table 4.7. Sometimes there are between 1-100 outliers whose errors are so great that they do not shift sigma circles. These outliers occur naturally in normally distributed data, such as this data, where the observed data point can be up to twice the standard deviation with the possibility of the point being up to three times the standard deviation.[3] With these outliers removed these results resemble the previous Case 2 simulations of Figure 4.5. In practice, a mobility solution would compare the current position to an estimated position to calculate the error. With the implementation of a state observer, the estimator works to smooth out anomalous spikes in data. As such, this anomalous point would likely be ignored. This will be further explained in a later chapter.

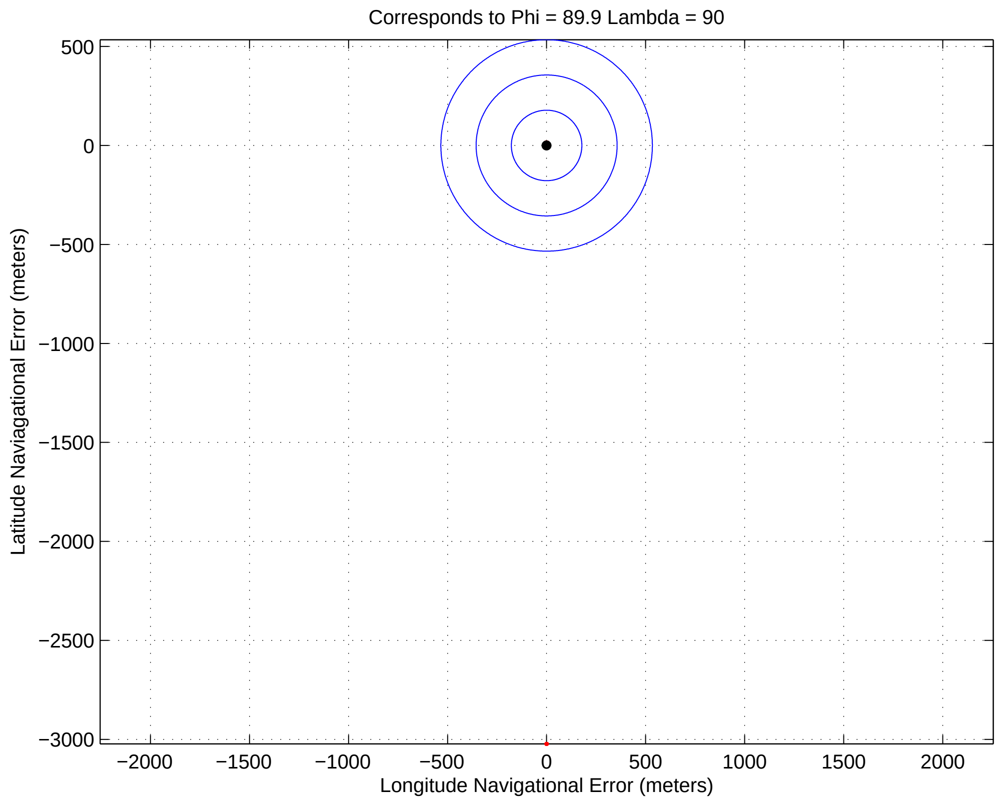

Figure 4.6: Case 2 (Accelerometer Measurement Noise) - Latitude 89.9° Longitude 90°

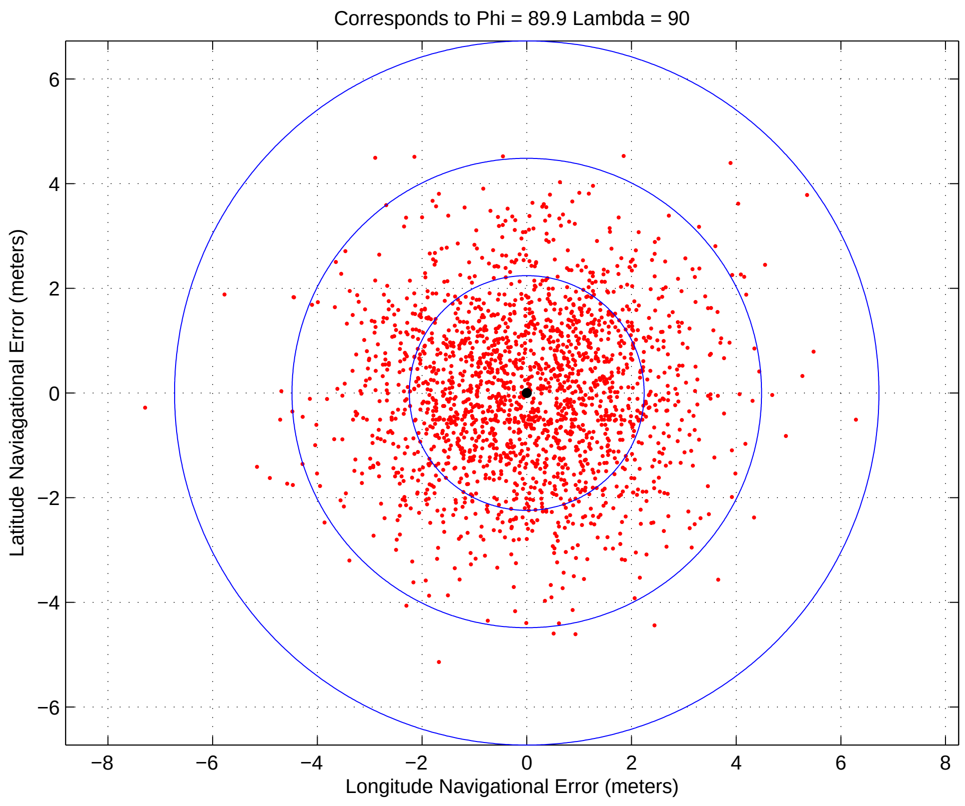

Figure 4.7: Case 2 (Outliers Removed) - Latitude 89.9° Longitude 90°

Even at this extreme latitude of 89.9° Figure 4.8 still follows the same pattern as other simulations at the worst case scenario (Example Figure 4.4).

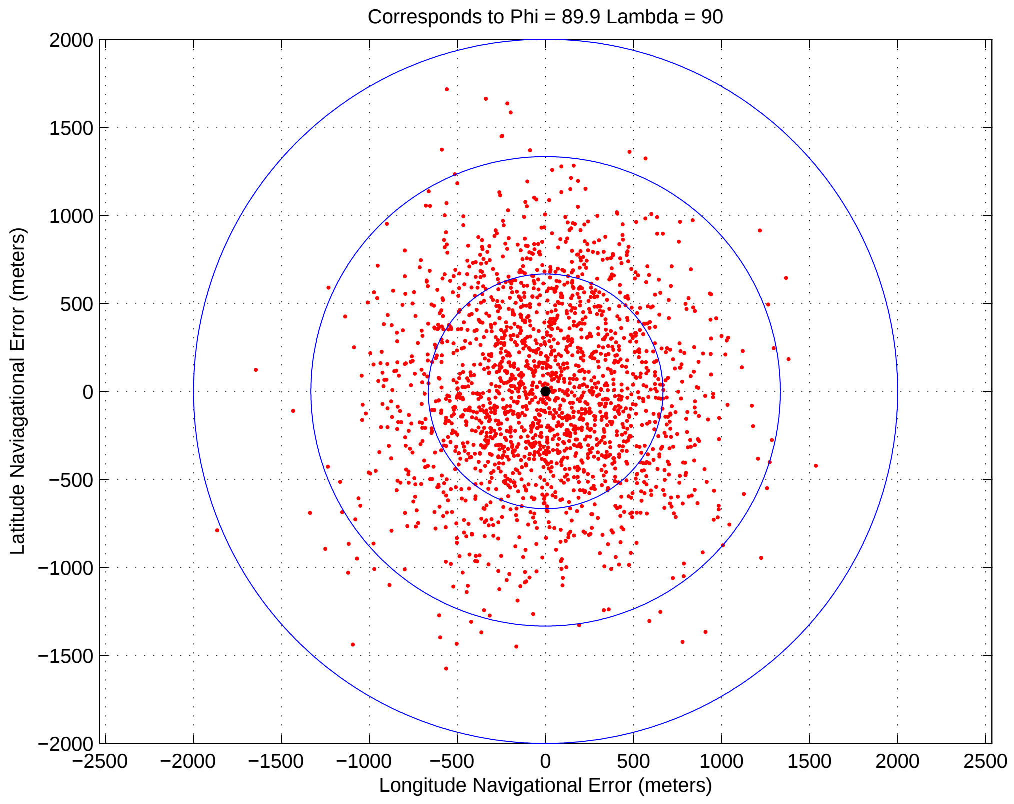

Figure 4.8: Case 3 (Worst Case Noise) - Latitude 89.9° Longitude 90°

4.5.3 Test Case - Latitude 90 Longitude 90

	Location		Sensors Errors		
	Latitude	Longitude	Star Tracker Alignment	Accelerometer Alignment	Accelerometer Noise
Case 1	90	90	0	0	0
Case 2	90	90	0	0	$1 * e^{-6}$
Case 3	90	90	60 arcsec	60 arcsec	$1 * e^{-6}$

Table 4.8: Latitude 90 and Longitude 90 - Test Case Criterion

This final simulation focuses on the most extreme case, that of 90° north or south latitude. This is to simulate being exactly at the poles of the Lunar body.

As can be seen in Figure 4.10 the same visual can be seen as in the previous cases with no noise, and navigational errors on the order of magnitude of 10^{-11}m, still showing small enough error to be considered numerical error.

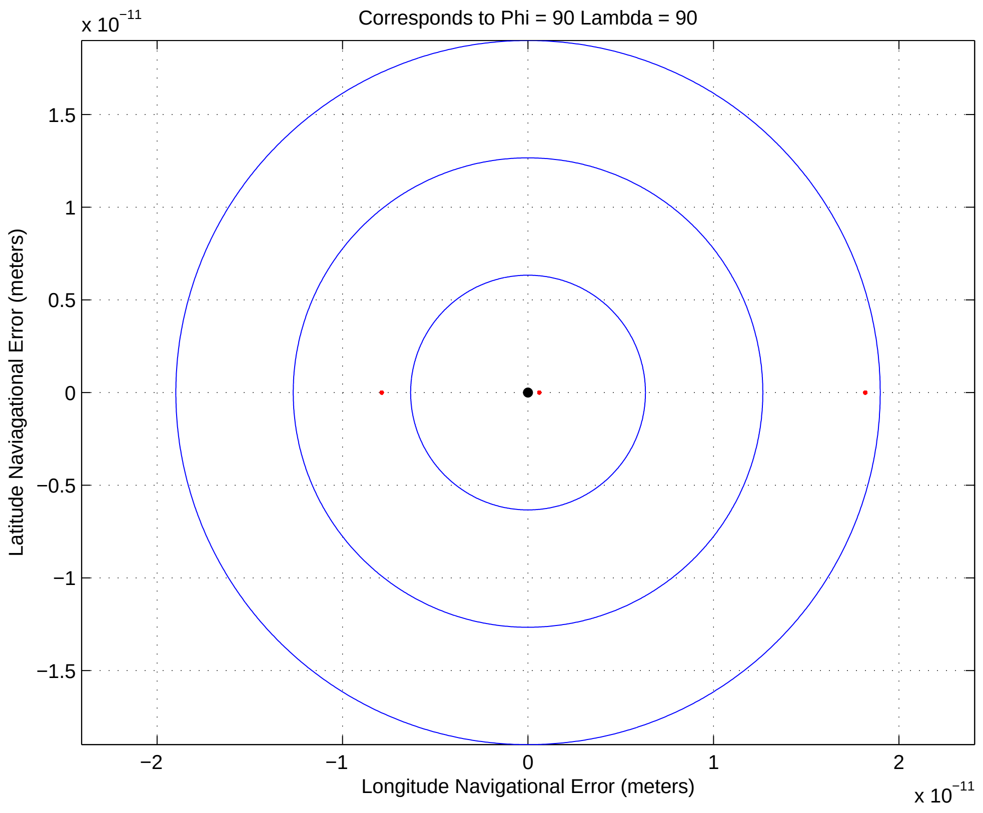

Figure 4.9: Case 1 (No Noise) - Latitude 90° Longitude 90°

Once noise is added to the systems it is possible to see an interesting effect. In Figure 4.10 there is no longitudinal error. This is because all longitudinal lines converge to a point at the poles as seen in Figure 4.11. Thus, the polar regions navigational angle error (λ_e) can be large, which could still translate to a small navigational error. This is due to how close the lines of longitude exist to each other. A small location error can translate to many degrees of latitude away from a desired location. Again it can be observed that with only accelerometer and star tracker measurement error, the total navigational error is very small, extending only 4 meters.

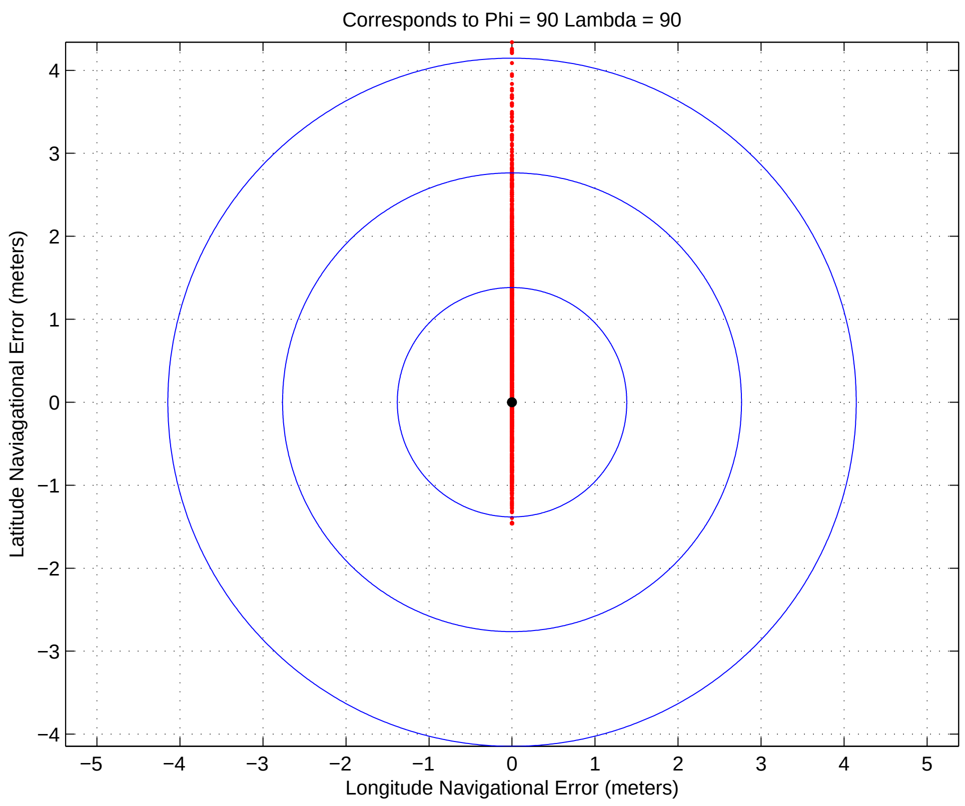

Figure 4.10: Case 2 (Accelerometer Measurement Noise) - Latitude 90° Longitude 90°

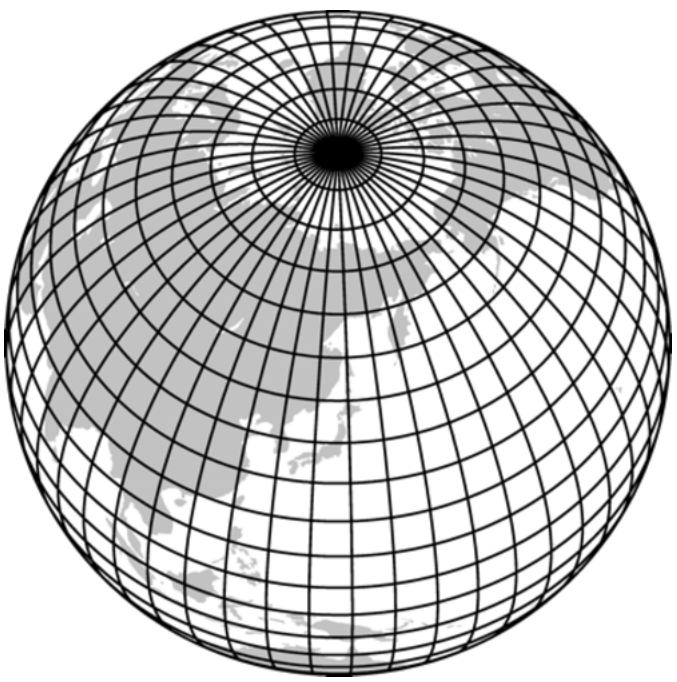

Figure 4.11: Latitude and Longitude Line Convergence [30]

Similar results can be seen in Figure 4.12. But again using the “worst case” noise scenario the error jumps to 1500m with some samples occurring over 1500m. (Recall that 2500m is at the upper limit of visibility on the Lunar surface.)

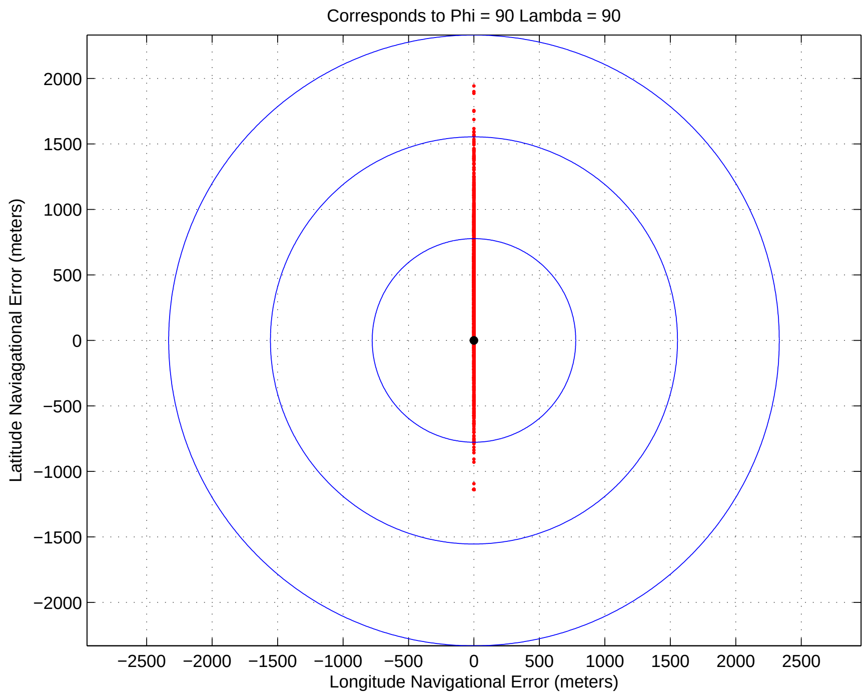

Figure 4.12: Case 3 (Worst Case Noise) - Latitude 90° Longitude 90°

4.6 Conclusions of Monte Carlo Analysis

Monte Carlo analysis shows that over a large sampling population (n = 2000+), there is a larger variance of possible locations based on the different levels of sensor noise and alignment error. CelNav shows a gaussian distribution of results for all levels of noise, excluding no noise. The results shown in this chapter are very promising considering that analyses are performed for a worst case un-calibrated scenario. Thus, even in the worst case scenario, minus a full sensor failure, CelNav appears to perform acceptably, within the upper limit of line of sight on the Lunar surface.

Chapter V

Preliminary Experimental Validation of CelNav - SkyScout

In this chapter preliminary verification of CelNav using hardware is performed using the SkyScout personal astronomy system produced by Celestron. The SkyScout is a hand-held device and is commercially available. This home astronomy device has the ability to detect its pointing direction and the celestial body the user is currently viewing, as well as the ability to give directions as to how to orient the SkyScout to view a chosen celestial body.

The SkyScout contains an internal inertial measurement unit (IMU) as well as a GPS. This information is available in single sample form from a user menu but requires using proprietary software from the vendor. Using this information it is possible to extract raw data from the SkyScout. SkyScout outputs several variables such as GPS coordinates, time, altitude, azimuth, rotation, right ascension, and declination. Using this information it is possible to construct all the necessary matrices (startracker (Δ) and Accelerometer (Γ) to verify the CelNav algorithm.

5.1 Developing CelNav Matrices

In this section, the Celestron SkyScout data is incorporated into the CelNav algorithm to confirm the algorithm's legitimacy. It should be noted that the Skyscout is used only as a "stand alone" sensor platform supplying the measurement data necessary for the CelNav algorithm. The development of both the startracker (Delta) and accelerometer (Gamma) was discussed earlier in Chapter 2. Here only the information supplied from the Skyscout to populate matrices, Δ and Γ, are examined against the "true" latitude and longitude extracted from the internal GPS unit.

5.1.1 Converting Julian Time to Earth-Center-Earth-Fixed

Sidereal time is an astronomical timekeeping system that allows astronomers to determine where to look for certain celestial bodies in the night's sky. This is due to the fact that if one is able to determine an object's location in the night's sky at a given time, on the next night the celestial object should be in the same location. Such information can be obtained from U.S. Navel Observatory[18]. Sidereal time can then be converted into Greenwich Hour Angle (GHA), which is a measure of the angle (in degrees) from any point on Earth to the prime meridian.

In order to accurately transform from the Earth-Center-Inertial (ECI) coordinate system, (which do not rotate with Earth), to that of the Earth-Center-Earth-Fixed (ECEF) coordinate system, which does rotate with Earth, the current number of days in Julian time must first be determined. Julian Time can be obtained via a Julian calender or calculated by the number of days that have passed since January 1, 4713 BC Greenwich noon.[17] This can then be converted into

Sidereal Time.

The GHA can be easily derived from Sidereal time by taking the remainder of:

$$\text{GHA} = (\text{Sidreal Time} + 450)/360 \tag{5.1}$$

this angle can then be converted into the rotation matrix Ω_{ECI}^{ECEF}, which describes rotation from the Earth-Center-Inertial coordinate system to that of the Earth-Center-Earth-Fixed coordinate system such that:

$$\Omega_{ECI}^{ECEF} = \begin{bmatrix} \cos(GHA) & \sin(GHA) & 0 \\ -\sin(GHA) & -\cos(GHA) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

5.2 Constructing Startracker (Δ) and Accelerometer (Γ)

The SkyScout is used to extract the raw data, startracker (Δ) and Accelerometer (Γ), necessary to verify the CelNav algorithm. The initial results are tested in Greenbelt, Maryland (latitude 39.0°N and long 76.89°W). All the tests, 28 included, were performed outside in a clearing, with the SkyScout mounted to a tripod for stability.[22] Each test was performed in the same location at the same altitude, only the heading and the angle of the Skyscout are modified.[22]

Extracting Δ and Γ is easily achieved since the SkyScout has the capacity to calculate the local latitude and longitude from internal sensors. The resulting Γ and Δ matrices can be inserted into the traditional CelNav equation for extracting latitude and longitude:

$$\Phi = [\Gamma^{-1}][\text{Ma}^{-1}][\text{Mc}][\Delta][\Omega^{-1}] \tag{5.3}$$

These results can then be compared to the "True" (experimental) latitude and longitude from GPS data.

Gamma is given as:

$$\Gamma = [M_a][ST] \tag{5.4}$$

where M_a is the accelerometer misalignment matrix and [ST] is the slope tilt matrix from Equation. (2.37). M_a represents an orthogonal misalignment. [ST] is constructed from the altitude and azimuth data such that:

$$[ST] = \begin{bmatrix} \cos(Altitude) & \sin(Altitude)*\sin(Azimuth) & -\sin(Altitude)*\cos(Azimuth) \\ 0 & \cos(Azimuth) & \sin(Azimuth) \\ \sin(Altitude) & -\cos(Altitude)*\sin(Azimuth) & \cos(Altitude)*\cos(Azimuth) \end{bmatrix} \tag{5.5}$$

Delta is given, such that:

$$\Delta = [\mathrm{M}_c]^{-1}[\mathrm{STH}][\mathrm{U}][\mathrm{LL}][\Omega] \tag{5.6}$$

where [STH] is the slope, tilt, and heading matrix as defined as:

$$[\mathrm{STH}] = ([\mathrm{ST}(\alpha,\beta)][\mathrm{H}(\epsilon)])^T \tag{5.7}$$

$$[\mathrm{STH}]^T = \left(\begin{bmatrix} \cos(\beta) & \sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) \\ 0 & \cos(\alpha) & 0 \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\epsilon) & \sin(\epsilon) & 0 \\ -\sin(\epsilon) & \cos(\epsilon) & 0 \\ 0 & 0 & 1 \end{bmatrix} \right)^T \tag{5.8}$$

As seen in Chapter 2 LL is the latitude (ϕ) and longitude (λ) matrix as defined as:

$$\begin{bmatrix} -\sin(\lambda) & \cos(\lambda) & 0 \\ -\cos(\lambda)\sin(\phi) & \sin(\lambda)\sin(\phi) & \cos(\phi) \\ \cos(\lambda)\cos(\phi) & \sin(\lambda)\cos(\phi) & \sin(\phi) \end{bmatrix} \tag{5.9}$$

Again U can be defined as:

$$U = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{5.10}$$

Finally $[\mathrm{M}_c]^{-1}$ and $[\Omega]$ are the transformation matrix relating coordinate frame star tracker with respect to coordinate frame Body and the planetary body's gravity model respectively.

When these matrices are combined they form:

$$[\text{STH}]_1 = \begin{pmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) \\ \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) \\ -\sin(\beta)\cos(\alpha) \end{pmatrix} \tag{5.11}$$

$$[\text{STH}]_2 = \begin{pmatrix} -\cos(\alpha)\sin(\epsilon) \\ \cos(\alpha)\cos(\epsilon) \\ \sin(\alpha) \end{pmatrix} \tag{5.12}$$

$$[\text{STH}]_3 = \begin{pmatrix} -\sin(\beta)\cos(\alpha) \\ \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) \\ \cos(\beta)\cos(\alpha) \end{pmatrix} \tag{5.13}$$

$$= \begin{bmatrix} [\text{STH}]_1 & [\text{STH}]_2 & [\text{STH}]_3 \end{bmatrix} \tag{5.14}$$

5.3 Results

Test cases are performed for CelNav algorithm verification. Without loss of generality, two randomly selected test case studies, test case 14 and 28 are represented here. These two cases are typical tests that reflect overall what is seen using the Skyscout as the main sensor for the CelNav algorithm.

It is assumed that a 3σ max of 60 arcsec noise represents the maximum misalignment in the accelerometer and star camera. Therefore, this noise is artificially introduced into the raw data. Three different situations are examined: no noise, expected noise, and 60 arcsec noise. Note all noise are evenly distributed about the *x*-*y*-*z* axes of each sensor. In Table 5.1 the “true” values of latitude, longitude and heading from the SkyScout’s GPS can be found for Case 14. Additional sensor noise is not applied due to the use of real sensor data from the SkyScout.

5.3.1 Case 14

Case 14 is performed at the NASA Goddard Space Flight Center (Greenbelt, Maryland)[22]. It is performed on an open structure with a clear view of the sky. The "true" values for the first test is provided in Table 5.1.

Truth Latitude	**Truth Longitude**	**Heading**
76.859°N	38.805°W	2.48°N

Table 5.1: Skyscout Truth Values Case 14

These values correspond to, within the margin of error of a standard GPS unit, the location of where the test is performed. It is possible to artificially corrupt the alignment matrices after the data is initially collected. Through manipulation of the alignment matrices it is possible to add error to each individual axis (x-y-z). The first test case, as seen in Table 5.2, has no added misalignment error.

As can be seen here, errors are insignificant. This test shows that with low error measurements CelNav is able to return accurate data. Navigational error can be calculated as:

$$\begin{aligned}\text{Position Error} = \; & \arccos(\cos(RA_{SS} * \tfrac{\Pi}{180}) * \cos(Dec_{SS} * \tfrac{\Pi}{180}) * \\ & \cos(RA_{CN} * \tfrac{\Pi}{180}) * \cos(Dec_{CN} * \tfrac{\Pi}{180}) + \\ & \sin(RA_{SS} * \tfrac{\Pi}{180}) * \cos(Dec_{SS} * \tfrac{\Pi}{180}) * \\ & \sin(RA_{CN} * \tfrac{\Pi}{180}) * \cos(Dec_{CN} * \tfrac{\Pi}{180})\end{aligned} \tag{5.15}$$

Where, RA$_{SS}$, is right ascension as determined by the skyscout, RA$_{CN}$, is the right ascension as determined by CelNav, Dec$_{SS}$, is declination from the SkyScout measurement, and Dec$_{CN}$, is the declination from CelNav.

The next test case is with added misalignment error, 20 arcsec of noise

	Trial 1 - 0 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	76.858°N	0	0
Longitude	38.805°W	0	
Heading	2.47°N	0	

Table 5.2: Skyscout Case 14 Trial 1 - 0 Arcsec Error

is chosen to represent expected levels of measurement noise. As can be seen in Table 5.3 the misalignment noise adds -0.007 degrees of error and 566.25m in navigational error. This shows that errors as small as 0.007 degrees result in significant navigational errors.

	Trial 3 - 20 Arcsec Error		
	Location	Angle Error(Degree)	Navigational Error (meter)
Latitude	76.866°N	-0.007	566.25
Longitude	38.810°W	0.007	
Heading	2.48°N	0	

Table 5.3: SkyScout Case 14 Trial 2 - 20 Arcsec Error

The next trial incorporates what is considered as a high amount of misalignment. As can be seen in Table 5.4 misalignment adds -0.021 and 0.018 degrees of error and 1571.15m in navigational error. Here it can be seen that the larger the misalignment, the greater the resulting latitude/longitude and navigational error.

	Trial 3 - 60 Arcsec Error		
	Location	Angle Error(Degree)	Navigational Error(meter)
Latitude	76.879°N	-0.021	1571.15
Longitude	38.822°W	0.018	
Heading	2.47°N	-0.003	

Table 5.4: Skyscout Case 14 Trial 3 - 60 Arcsec Error

5.3.2 Case 28

The following tables show a test case and at the same location as in Case 14 but with a different heading. The results are very similar to results previously discussed.(Overall noise levels are held constant.)

True Latitude	**True Longitude**	**Heading**
78.86°N	38.30°W	90.69°E

Table 5.5: Skyscout Truth Values Case 28

	Trial 1 - 0 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	78.858°N	0	0
Longitude	38.805°W	0	
Heading	90.69°E	0	

Table 5.6: Skyscout Case 28 Trial 1 - 0 Arcsec Error

	Trial 2 - 20 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	76.850°N	0.007	566.14
Longitude	38.811°W	0.006	
Heading	90.69°E	0	

Table 5.7: Skyscout Case 28 Trial 2 - 20 Arcsec Error

	Trial 3 - 60 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	76.836°N	0.022	1569.78
Longitude	38.822°W	0.017	
Heading	90.69°E	0	

Table 5.8: Skyscout Case 28 Trial 3 - 60 Arcsec Error

In examining Case 28 the results are quite comparable to those discussed in Case 14. As can be seen from preliminary experimental data, CelNav results show negligible latitude and longitude error using the GPS information when no misalignment exists. With additional misalignment error, and sensor noise held constant, even greater navigational errors result. Again remember, as little as 0.001 degree error translating into large navigational errors.

Chapter VI

Simulation Model

The analytical rover model used in this research is that of Sun et al.[8], which was originally a wheeled vehicle model designed to work with a Particle Swarm Optimizer application. This rover model is generic enough to be modified to fit the needs and scope of this research.

In this research, a rover is controlled via an observer-based controller. The observers use CelNav data for feedback updates. As the goal of this research is to analyze and compare the efficiency of the applied estimators/filters using CelNav feedback and not the controller, a simple PID controller is used without loss of generality.

6.1 Rover Model

In this model, for simplicity, a two-dimensional system is discussed, although in practical application a three-dimensional model would more closely represent an extra-planetary body. A brief description of vehicle dynamics as they appear in Sun et al. is discussed here, after examining how the model is modified to meet the needs of the CelNav algorithm.

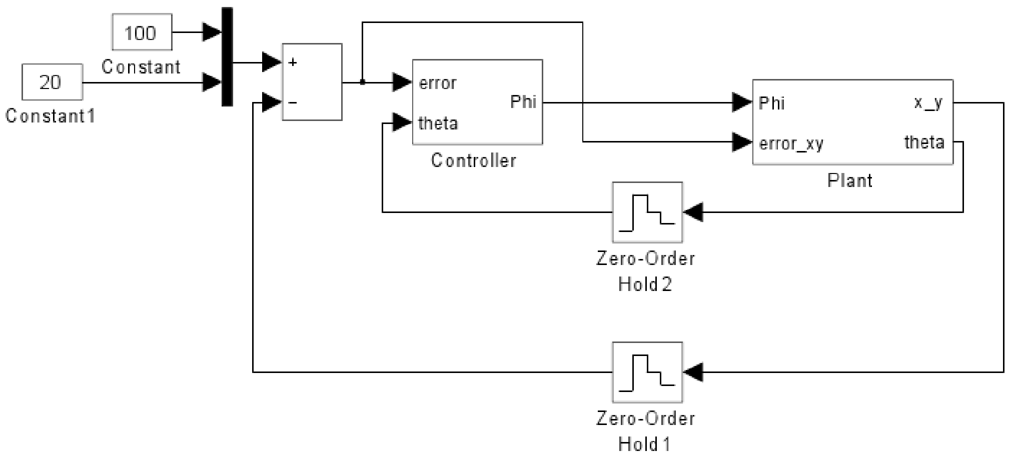

Figure 6.1: Simulation Model

First an overview of the vehicle dynamics as they appear in Figure 6.1 here in Table 6.1.

x	Vehicles position on the x axis
y	Vehicles position on the y axis
θ	angle between the velocity direction and the inertial X-Axis (Heading)
l	length of the vehicles wheels front to back
ϕ	steering angle of front wheels
$\mathbf{q}_1$	error term of the degree variation of front wheels
$\mathbf{q}_2$	error term of the acceleration variation of the vehicle

Table 6.1: Variable from Rover Model in Sun et al.

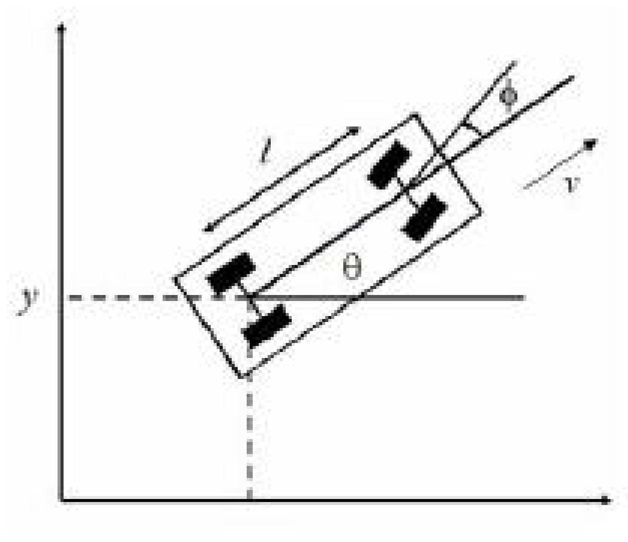

Figure 6.2: Rover Model [8]

As seen above, the vehicle's coordinate frame is given in the body coordinate system. This is different than how it will be discussed later, where it will be converted from inertial to body-centered coordinates. Here, using body coordinates is acceptable due to the fact that only a local system is being simulated.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{a} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{l}\tan(\phi) \\ a \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{6.1}$$

For this application the input variables q_1 and q_2 are removed and replaced with a ϕ correction term (to be discussed further in the next section). The $\dot{v}$, $\dot{a}$, and $\dot{\phi}$ terms are also removed in order to further simplify the system, leaving only the terms found in Equation 6.2.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{l}\tan(\phi) \end{bmatrix} \tag{6.2}$$

Velocity is now a fixed constant 0.1 $\frac{m}{s}$. The velocity term is found in Figure 6.3 under the block labeled “velocity” This block controls the system velocity based on x and y error. When this error decreases to within a predetermined threshold the velocity is set to zero, signaling that the vehicle has reached its destination. The current system is tuned so that when the error of x and y reaches 0.1 meters from is its intended destination the velocity is disabled.

Additional modifications are made to effect a more practical model. Saturation terms are placed on θ, x, and y due to their limited nature. For this system there is a limit to how much the vehicle is allowed to turn at each time step, as well as how far the vehicle is allowed to travel in the x and y direction. The system is such that the x and y displacements are limited to 180° to -180° latitude and 90° to -90° longitude, respectively, to simulate longitudinal and latitudinal limitations.

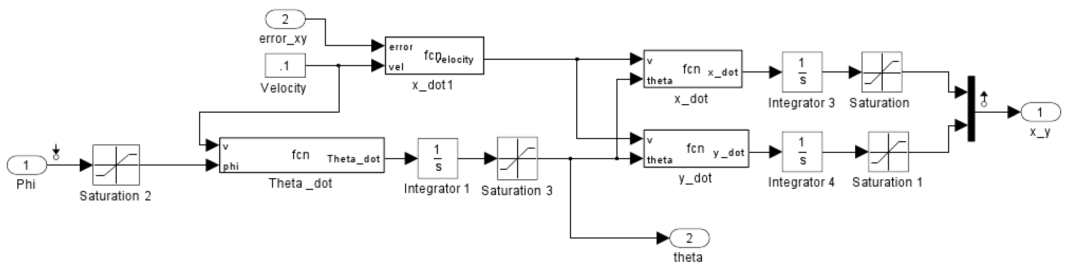

Figure 6.3: General Model for Rover

6.2 Proportional-Integral-Derivative (PID) Controller

It is important to reiterate that the actual controller is not the focus of this research. Rather, the focus is on the platform for testing estimators using CelNav. A simple PID controller is developed to replace the original Input-Output Linerization control applied in Sun et al. Stability is of prime importance since reaction speed is mostly dependent upon the reaction speed of the estimators.

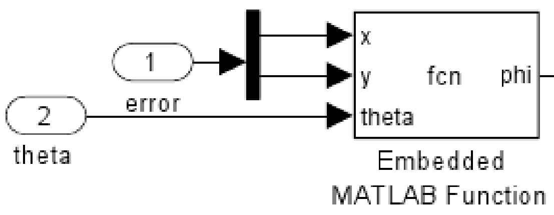

Figure 6.4: Controller Model

The control, as stated above, is a simple Proportional-Integral-Derivative (PID) controller. Again, the controller is not of interest in this research; only the performance of the three observers are to be analyzed.

For this example only the position error, x (Latitude) and y (Longitude), are calculated as can be seen in Figure 6.3. Only heading (θ) is taken into account for feedback due to the added complexity to the PID controller. These gains can be found in Table 6.2. Again the main goal for this controller is stability, with a focus on the analysis of the performance of the applied estimation techniques.

In order to control the system via its heading, it is necessary to calculate

P	**I**	**D**
1000	2450	0

Table 6.2: PID Gains

a heading correction factor (α). This is accomplished by feeding the heading as defined from the Plant in Figure 6.1 back into the controller. This correction factor, α, can be found in Equation 6.3:

$$\alpha = \tan^{-1}\left(\frac{y}{x}\right); \tag{6.3}$$

By taking the previous heading (θ) and subtracting if from α the corrected heading term can is realized, as seen in Equation 6.4:

$$\phi = \alpha - \theta; \tag{6.4}$$

Now that the correct heading term is known new x and y coordinates can be calculated. This is a simplified method for calculating a corrected heading, though it is possible to have a PID or other type of controller effectively manage this calculation.

Due to the difficulty of controlling this general rover model with a PID, as can be seen by the large gains in Table 6.2 and using a correction factor as the control input, there is a very large control effort, as seen in Figure 6.5. Attempts are made to reduce the magnitude of the control effort, but resulted in the inability to accurately control the system, sometime causing unstable conditions.

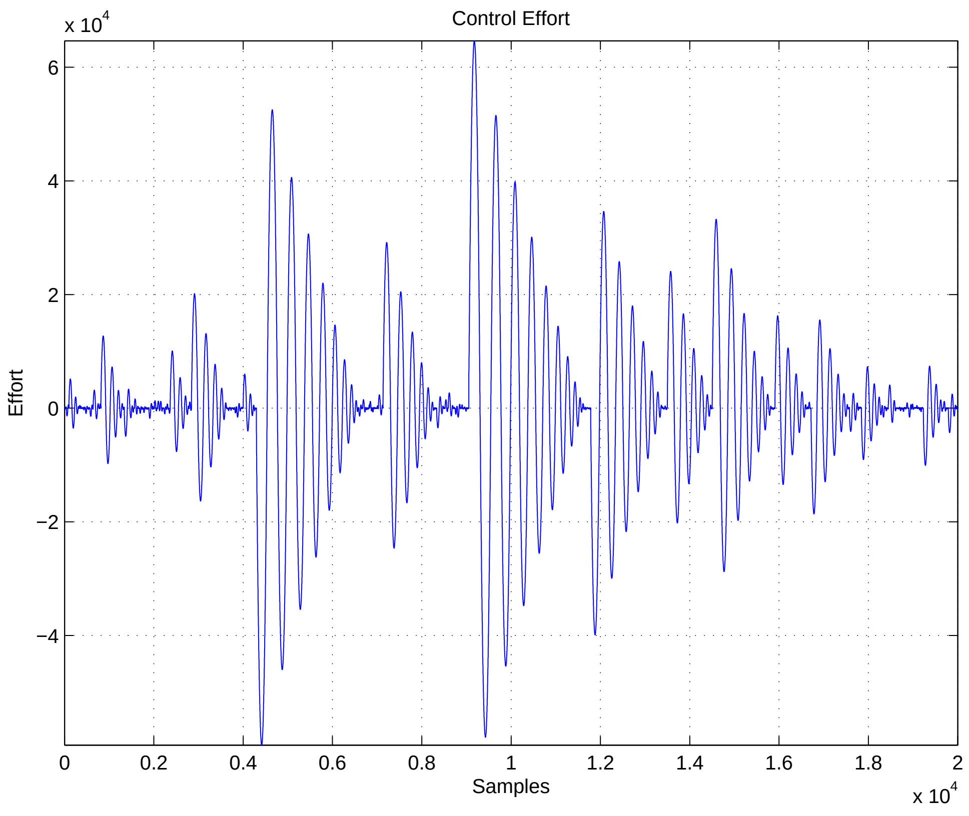

Figure 6.5: Control Effort

6.3 Results

For this simulation only accuracy and prevention of overshoot are deemed as the limiting factors. As can be seen in Figure 6.6 the rover moves to the desired location in almost a stepped fashion. The x value increases only when the y value achieves certain thresholds. The figure below shows that there is little to no overshoot which is what this simulation is trying to achieve and both the x and y error reach the desired boundary conditions within a few time steps.

Heading can also be seen in Figure 6.6, fluctuating between +4 degrees and -4 degrees. This coincides with the what is seen in the x and y plots in the same figure.

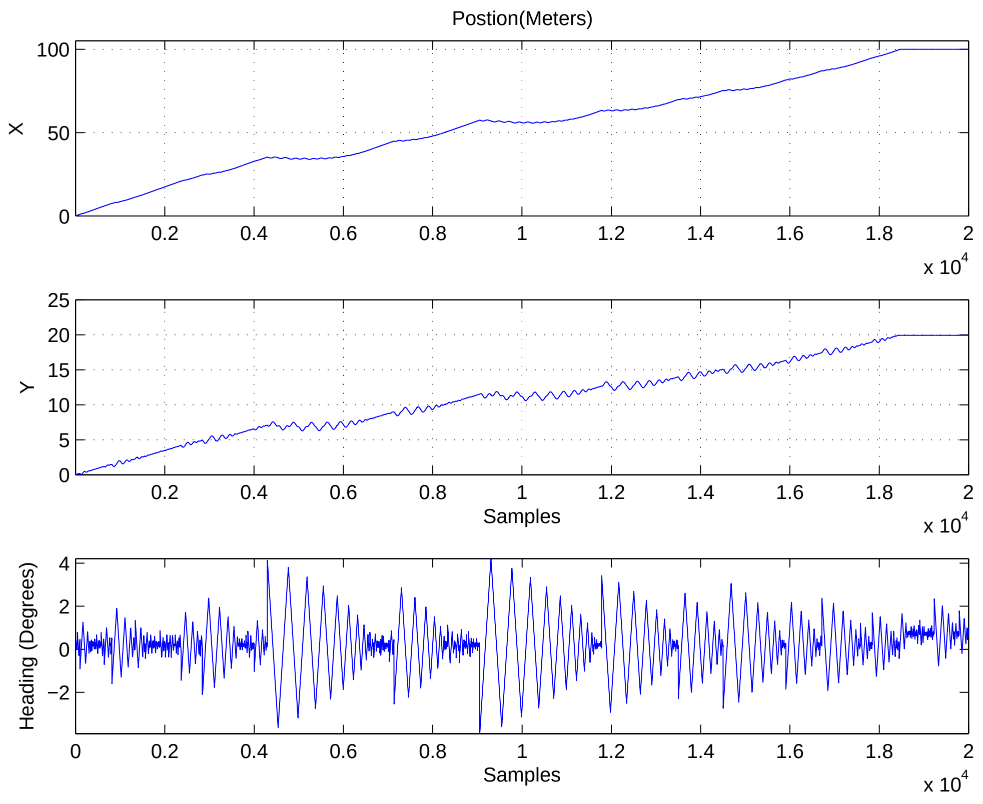

Figure 6.6: PID Results (X, Y, and Heading)

The rover's trajectory can be seen in Figure 6.7. When Figure 6.7 is compared to Figure 6.6 one may observe the same oscillations in the x vs y trajectory as in the heading. This is due to the rover system trying to arrive at the desired location as quickly as possible, which in this case happens to be almost in a straight line.

The error x and y error can be seen in Figure 6.8 it can be seen that the system does eventually reach 0 error after 2000 seconds, in much of the same fashion that the system reaches its destination.

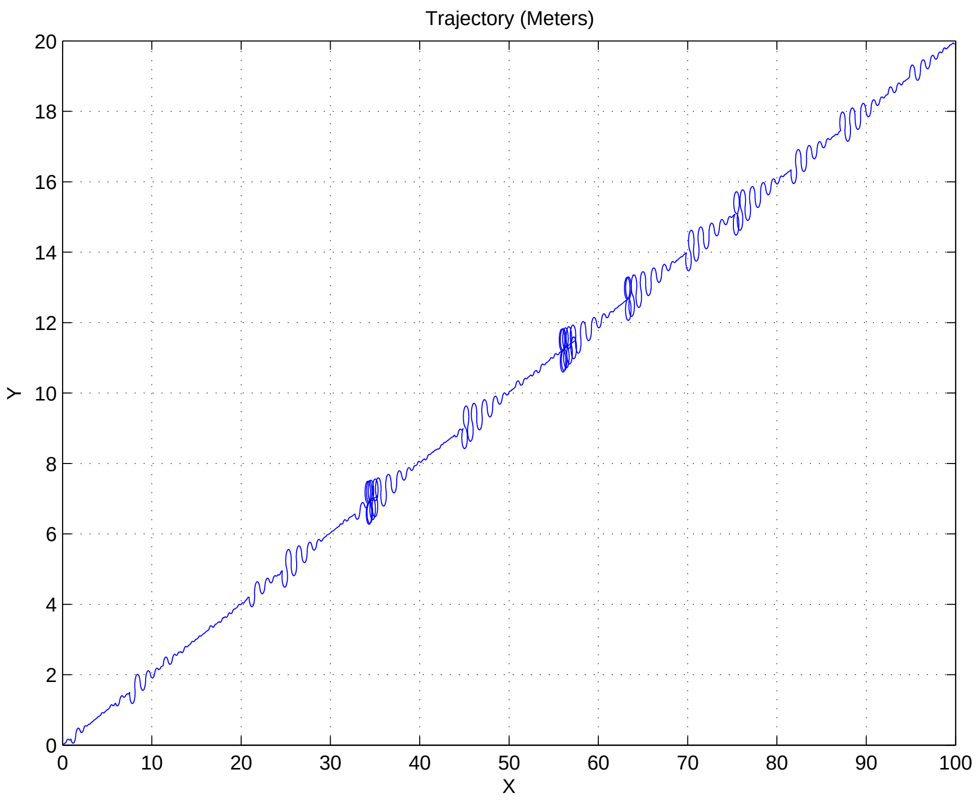

Figure 6.7: X and Y Trajectory

It can be seen in Figure 6.7 that the vehicle does not travel in a perfectly straight path. This is due to the nature of the controller, with the heading being determined by a correction factor rather than a PID or other such controller. It may be possible to further refine the controller to improve the results, but this is not the focus of this research. Also, the elimination of the higher order terms seem to have little effect on the vehicle model as a whole, which may lead to the speculation that even though the model is a generic 2-D model, it can accurately describe a four wheel, front turning vehicle. The model as described above is further compared to a tread vehicle in later chapters. It can also seem that the system may benefit from the use of an estimator to help dampen out some of the existing chattering motion.

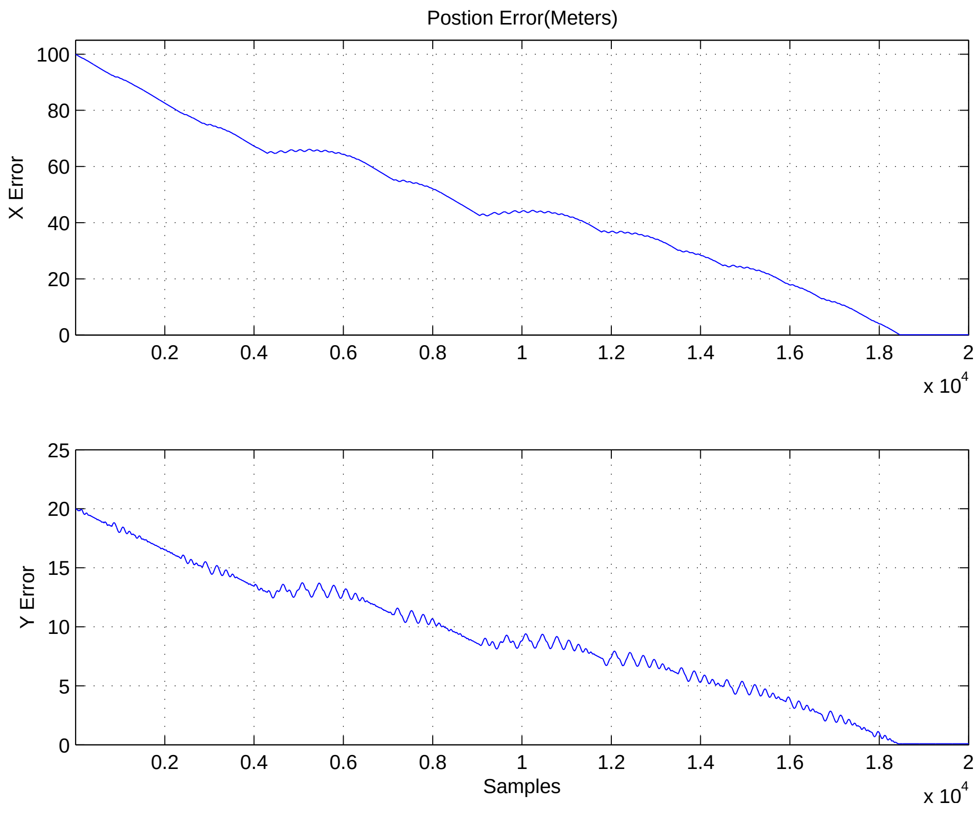

Figure 6.8: XY Error

Chapter VII

Simulation Results

This chapter focuses on analytical results from various simulation conditions using three different estimation techniques: Extended Kalman Filter, H-Infinity, and Sliding Mode Observer. Each of these techniques are analyzed individually. An overall comparison is provided in the next chapter. Each technique is analyzed with a focus on accuracy to true measurement, overall noise reduction and simulation efficiency. Additionally, two different analyses will be looked at, (1) the performance of each of the estimators running "outside of the loop" of the rover model, (2) the performance of each of the estimators running "in the loop" using CelNav as "sensor" feedback. This allows for a comparison and the performance of the three different estimators and how they react to possible erroneous and/or noisy CelNav output.

The controller of the nonlinear rover plant is highly dependent on the accuracy of measurement (or estimate) feedback. Thus, state estimators are needed to "clean" inaccurate sensor feedback data. The simplified version of the rover control system can be seen in Fig 7.1. This model has a PID controller which is updated using only one of the three available states (heading θ), and the nonlinear plant model shown in Equation 7.1. Only measurement noise is investigated in the following chapters, although alignment error has a significant effect on the CelNav algorithm.

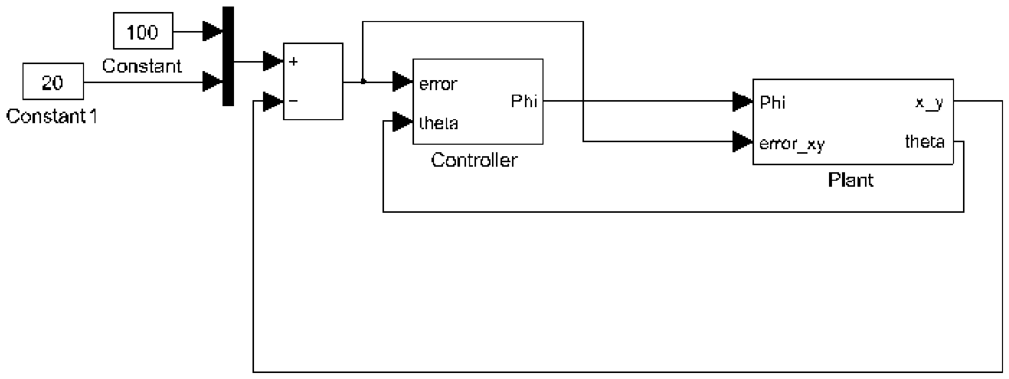

Figure 7.1: PID Feedback Control System

In Equation 7.1 is the rover plant model. Where v is the velocity of the rover, θ is the rover's heading, and l is the length of the rover.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cdot \cos\theta \\ v \cdot \sin\theta \\ \frac{v}{l} \cdot \tan\theta \end{bmatrix} \tag{7.1}$$

The values for the PID system are as follows in Table 7.1:

PID Gains	
P	1000
I	2450
D	0

Table 7.1: PID Gains

It is determined that the system needs to be able to react quickly to changes in location, which is the determining factor in selecting the proportional gain. The integral gain is chosen such that system will be quickly driven to the set point.

The initial conditions for the PID controlled rover system can been seen in Table 7.2. These conditions are the same for each of the estimators. The final

Initial Conditions	
Latitude	0
Longitude	0
Velocity	.1 $\frac{m}{s}$

Table 7.2: Rover Initial Conditions

conditions for the PID system can be found in Table 7.3:

Final Conditions	
Latitude	20
Longitude	100
Velocity	0 $\frac{m}{s}$

Table 7.3: Rover Desired Coordinates

The observers are tested in the simulated environment as measurement based control with the addition of the CelNav algorithm (i.e. open-loop estimation) to observe the algorithm performance under high noise environments.

The rover based autonomous navigation system is highly dependent on measurements with minimal noise residual. To accomplish this, four different estimation techniques are examined; Luenberger Observer, Extended Kalman Filter (EKF), Sliding Mode Observer (SMO), and H-Infinity (H-$_\infty$) filter. Each observer and its associated properties are discussed at length in the following sections.

This chapter will focus on the comparison of two levels of noise: expected noise levels and "worst case" levels. The reader should note that the goal is to test the validity of the EKF(and the other estimators) on the defined rover system model being used and not the performance of the controller. Two different simulations are run for each observer, (1) Expected Noise Levels and (2) "Worst Case" Noise Levels.

The estimators to be discussed below are to examine the ability for improved location determination of the CelNav algorithm by accounting for: 1. measurement noise, 2. process noise, and 3. other unknown uncertainties. Again, all noise is assumed to be Gaussian. Each section compares the estimators under the expected and "worst" case noise levels as stated above in three different formats: 1.) Trajectory plot which shows the path the "rover" travels to reach its destination, 2.) Error Plot (Meters) which shows the location error of the rover in meters, and 3.) Error Plot (Degrees), a plot that shows the location error in degrees from true latitude and longitude. Two different conditions are examined with respect to: observer-based control and measurement-based control.

7.1 Extended Kalman Filter

Optimal Estimation of the Extended Kalman Filter (EKF) works on the premise of returning the best statistical state estimate. As defined in Gelb [23] an optimal estimator is a computational algorithm that processes measurement to deduce the minimum error estimate of the state of a system by utilizing: 1.) knowledge of the system and measurement dynamics, 2.) assumed statistics of system noise and measurement errors, 3.) initial condition information.

This method, though, is sensitive to incorrect or erroneous plant and measurement models. The EKF can also be computationally expensive due to a required matrix inversion at each iteration. The EKF works by using a linearized model of a nonlinear system. This is accomplished by re-deriving a new state matrix at each new time step using information from the previous and current iteration. The EKF simulation model can be seen in Figure 7.2.

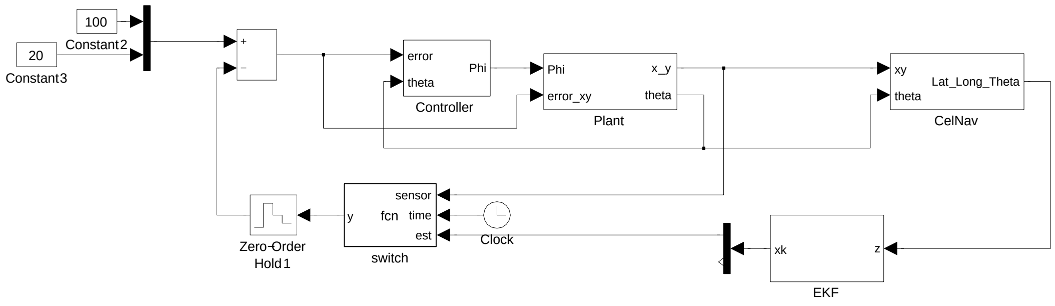

Figure 7.2: Extended Kalman Filter Simulation Model

7.1.1 EKF Governing Equations

The EKF is applied to general nonlinear system and measurement equations assumed of the following form:

$$\begin{aligned} x_k = &\quad F_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \\ y_k = &\quad H_k(x_k, \upsilon_k) \end{aligned} \tag{7.2}$$

The equations above contain both measurement and system noise terms, υ_k and w_k, respectively. Also contained above are x_k and x_{k-1} which are the current and previous state values, and u_k is the controller gain. Here y_k is defined at the sensor measurement. The following equation sets can be readily followed using those contained in Simon [8].

When computing the Kalman filter equations, two partial differential equations need to be calculated. First is the system partial with respect to x_{k-1}^+ as can be seen below:

$$F_{k-1} = \frac{\delta f_{k-1}}{\delta x}\bigg|_{\hat{x}_{k-1}^+} \tag{7.3}$$

This must be updated at each time step to account for new x^+ values. Next it is necessary to update the estimate covariance matrix:

$$P_{k-1} = F_{k-1}P_{k-1}^+F_{k-1}^T - L_{k-1}Q_{k-1}L_{k-1}^T \tag{7.4}$$

The state estimate is given as:

$$\hat{x}_k^- = f_{x-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) \tag{7.5}$$

The above calculations are performed using previous time-step information. The following equations are calculated using both current measurements as well as the above updates. First it is necessary to update a Kalman gain as follows:

$$K_k = P_k^- H_k^T (H_k P_K^- H_K^T + M_k R_k M_k^T)^{-1} \tag{7.6}$$

Here, H_k is the matrix used to define state parameters corresponding to measurement output and M_k is assumed to be 1 due to unknown measurement noise models. The final state estimate update law is as follows:

$$\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - h_k(x_k^-, 0)] \tag{7.7}$$

with the resulting estimation-error covariance matrix as:

$$P_k^+ = (I - K_k H_k) P_k^- \tag{7.8}$$

The EKF is updated at each time step. Due this constant update, a one sample time lag occurs, meaning the state estimates and the measurements lag by one sample time step. The process noise covariance , Q, and the measurement noise covariance matrix, R, in this thesis are given as follows:

$$Q = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.9}$$

$$R = \begin{bmatrix} 1000 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2500 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2500 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.10}$$

It is important to note that there is no cross correlation between the process noise and the measurement noise. Q is determined by performing the least squares method on the process noise, then further refining the values to account for unmodeled bias and noise. R is determined from the predicted sensor noise variance.

7.1.2 EKF Simulation Results

Results from the closed-loop EKF as applied to the nonlinear simulated system with "expected" noise of $1 * 10^{-4}$ magnitude are shown in Figure 7.4. A table of values can be found at the end of this section (Tables 7.4 and 7.5). Figure 7.3 shows the trajectory of the rover. The plot shows that the "rover" traverses the distance in a relatively straight line. There is a very slight wobble in the trajectory (seen more clearly in Figure 7.4). This wobble occurs because the difficulty the "rover" has to travel in a straight line due to the EKF estimate error.

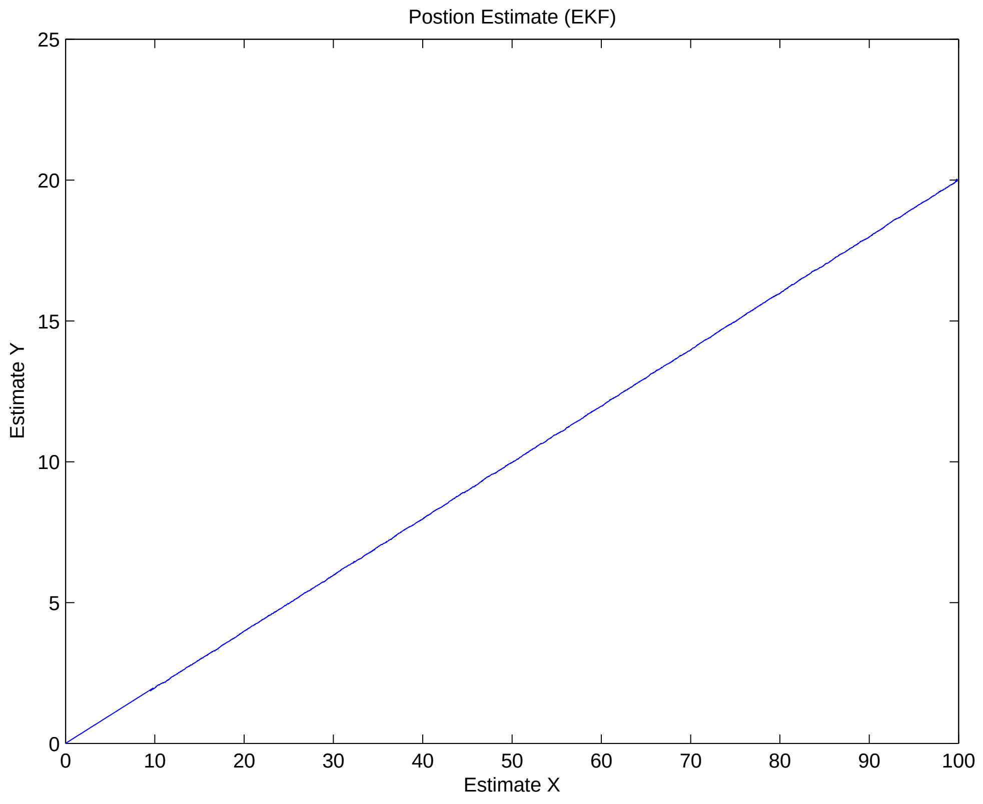

Figure 7.3: EKF - Estimated Trajectory

The position error (in degrees) is shown in Figure 7.4. Here it can be seen that there is an offset in both the x and y directions. This offset is due to the lag in the EKF being at least one time step behind. In Table 7.4 the mean of the position error is 0.282 degrees in latitude and 0.089 in longitude. Although this seems to be a relatively small error, (as described previously) this can translate into large position errors. Also important to note is the"tight" error band for Figure 7.4 where the rover travels in a relatively straight line.

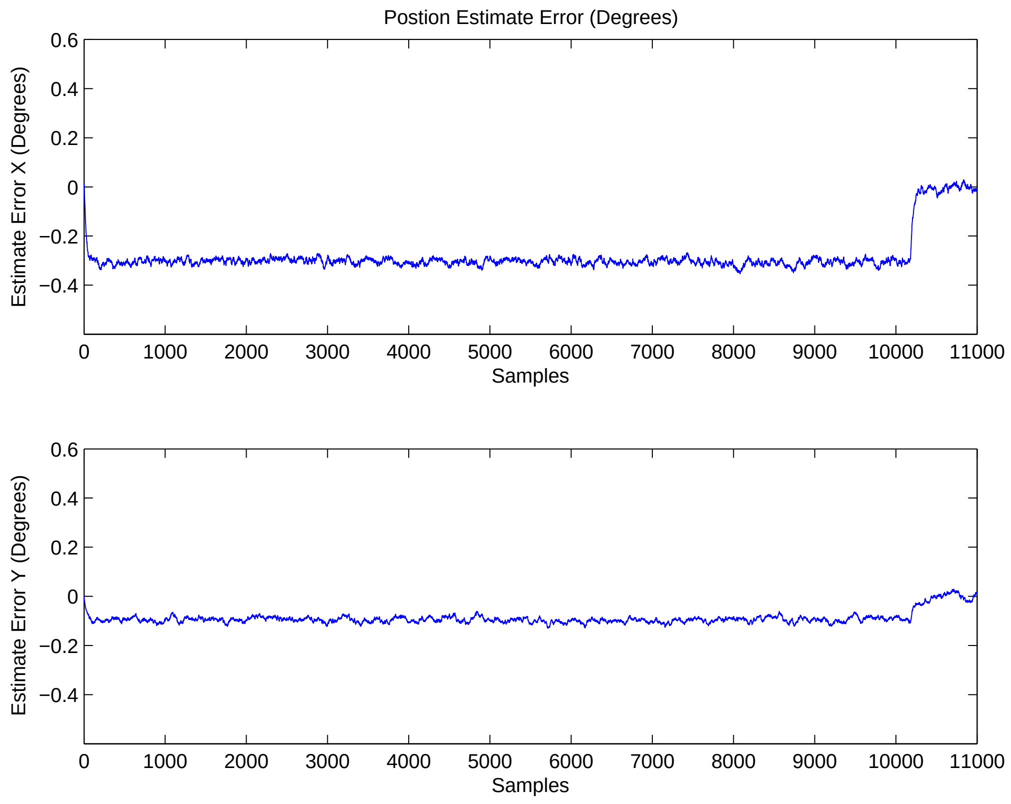

Figure 7.4: EKF - X and Y Estimate Error (Degrees) with Expected Noise

Mean [μ](Degrees)		Standard Deviation[σ] (Degrees)	
Latitude	Longitude	Latitude	Longitude
0.28238	0.08892	0.07800	0.02564

Table 7.4: EKF Mean [μ] and Standard Deviation[σ] for Expected Noise

Here in Figure 7.5 the error band for both the x and y directions is approximately 500 meters. As discussed in previous chapters navigational errors less than 2000 meters are acceptable, although smaller navigational errors are preferred. In Table 7.5 it can be seen that the maximum navigational error is 1300.10m with a standard deviation of 377.16m. The results are very reasonable when compared to that of the Monte Carlo simulation in Chapter 4, which shows the possibility for errors approaching 2000 meters. With the addition of an estimator it is possible to mitigate some of the instantaneous high error outliers that may have existed previously.

Max Navigational Error (Meters)		Standard Deviation[σ](Meters)	
Latitude	Longitude	Latitude	Longitude
1300.10	1027.70	377.16	314.43

Table 7.5: EKF Max Navigational Error and Standard Deviation[σ] for Expected Noise

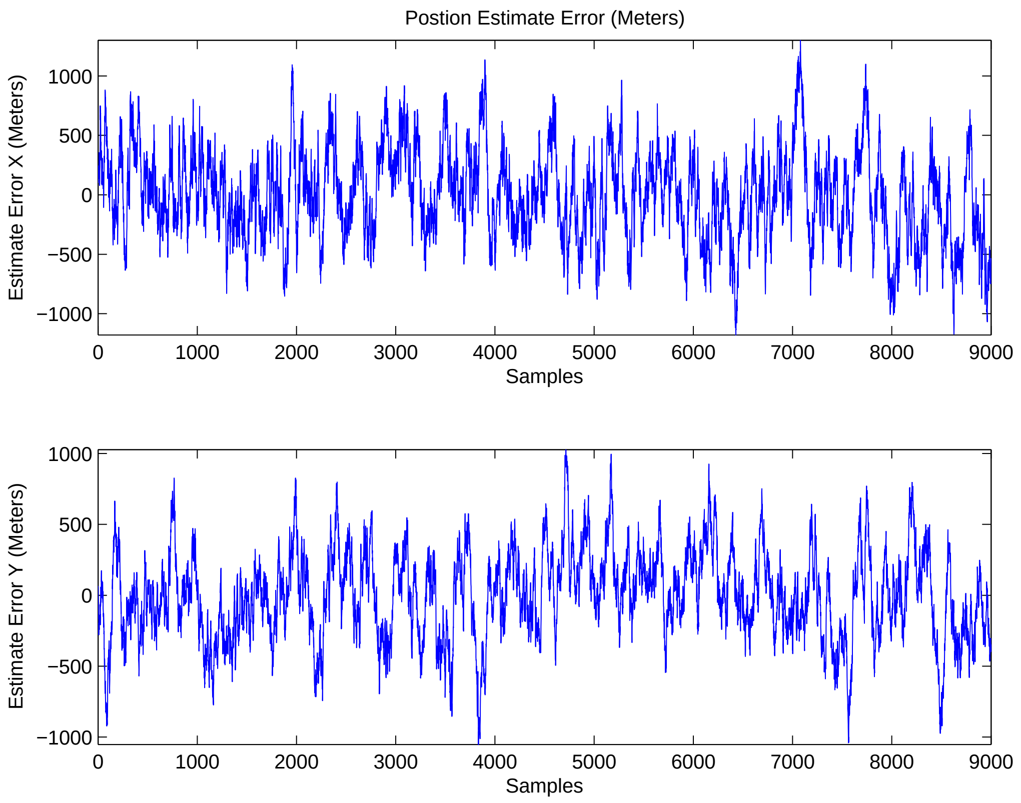

Figure 7.5: EKF - X and Y Estimate Error Expected Noise (Meters)

Results from the closed-loop EKF as applied to the nonlinear simulated system with "higher" levels of noise, $1 * 10^{-2}$ magnitude, are shown in Figure 7.7. A table of values can be found at the end of this section (Table 7.6 and Table 7.7). The trajectory of the "rover" is shown in Figure 7.6. The trajectory plot shows that the "rover" traverses the distance in a relatively straight line. There is a significantly greater wobble in the trajectory, which is due to the increased difficulty the "rover" has to travel in a straight line, because of the higher EKF state estimate errors.

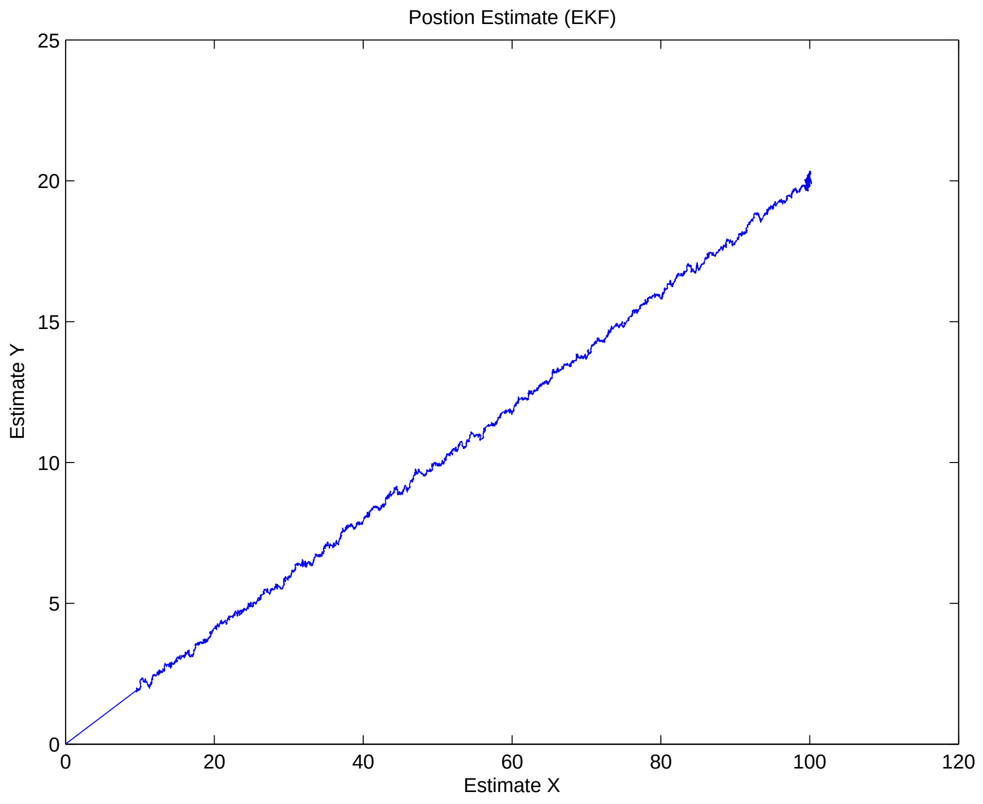

Figure 7.6: EKF - X and Y Estimate Trajectory with High Noise Levels

The position error can be seen in Figure 7.7. Here it can be seen that there is again an offset in both the x and y directions. Again, this is due to the lag in the EKF being at least one time step behind. In Table 7.6 the mean of the position error is 0.281 degrees in latitude and 0.0866 in longitude.

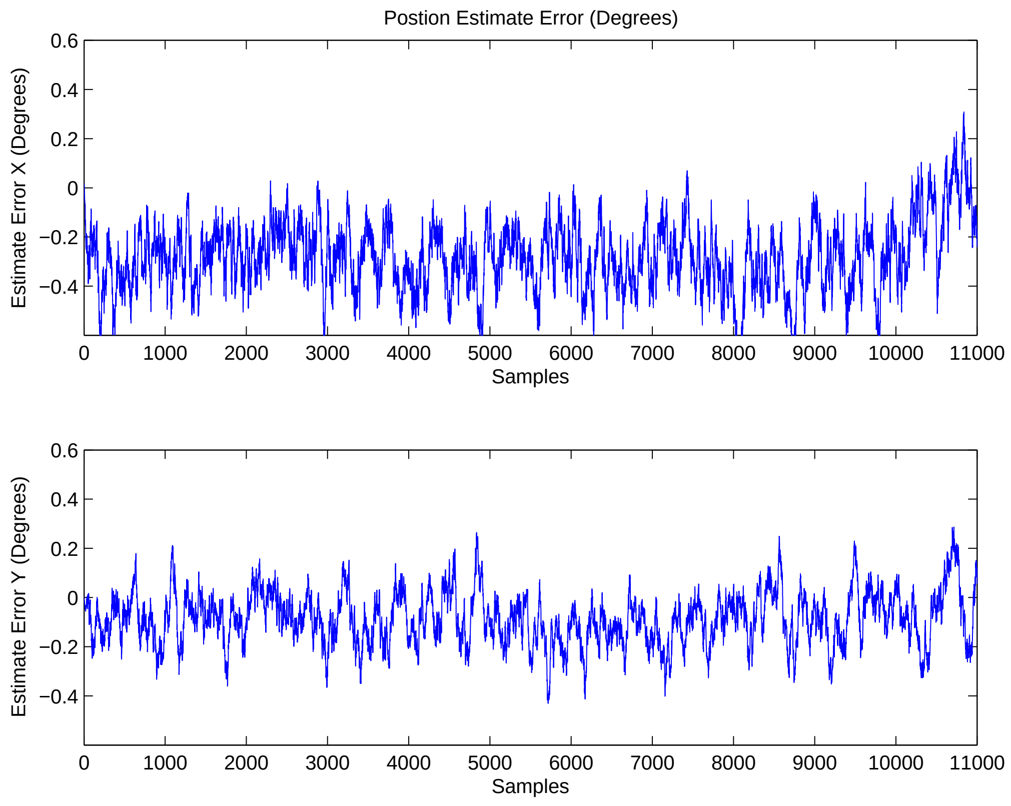

Figure 7.7: EKF - X and Y Estimate Error with "Worst" Noise (Degrees)

Mean [μ] (Degrees)		Standard Deviation[σ] (Degrees)	
Latitude	Longitude	Latitude	Longitude
0.28167	0.08667	0.14064	0.10675

Table 7.6: EKF Mean [μ] and Standard Deviation[σ] for "Worst" Noise

A small angular error can translate into hundreds or thousands of miles in position error. In Figure 7.8 the error bands for both the x and y directions are approximately 5000 meters. Again, navigational errors less that 2000 meters are acceptable, as stated previously. This navigational error of 5000 meters far exceeds the line-of-sight range of a rover or astronaut. This shows how important reducing the overall error is in extra-terrestrial navigation. In Table 7.7 it can be seen that the maximum navigational error is 14105.86m with a standard deviation of 3712.74m.

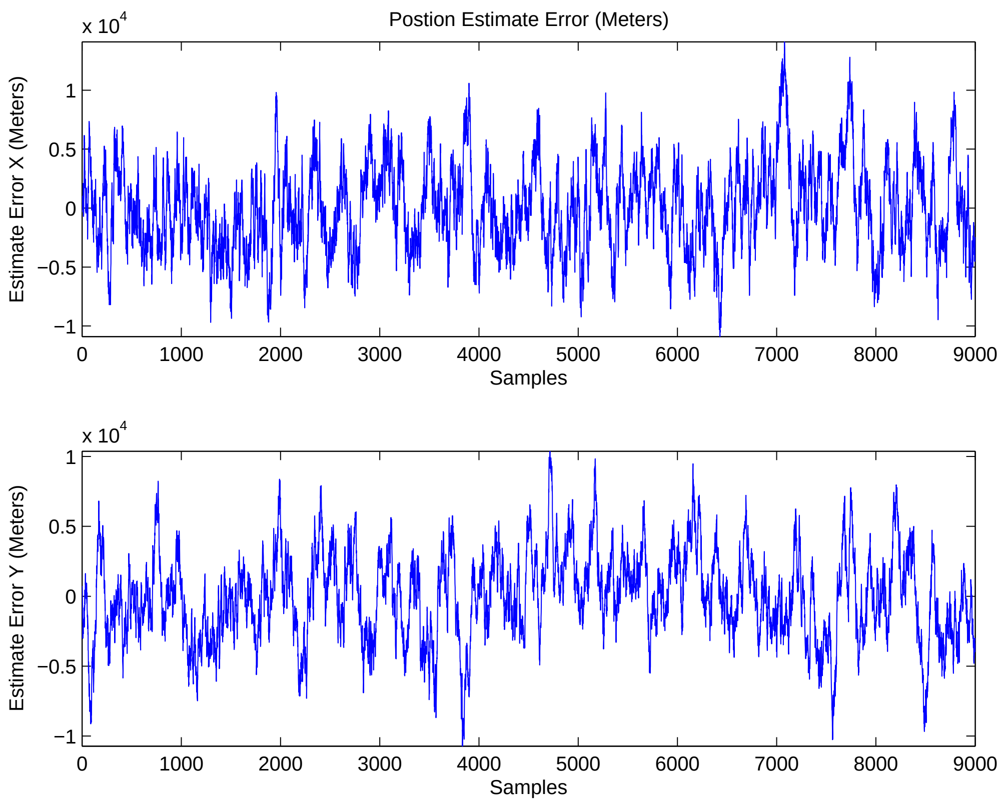

Figure 7.8: EKF - X and Y Estimate Error Higher Noise (Meters)

Max Navigational Error (Meters)		Standard Deviation[σ](Meters)	
Latitude	Longitude	Latitude	Longitude
14105.86	10376.66	3712.74	3146.43

Table 7.7: EKF Max Navigational Error and Standard Deviation [σ] with Higher Noise

7.2 H-Infinity ($\mathbf{H}_\infty$)

H-Infinity (H_∞) is very similar to that of the EKF method. H_∞ was created as a way to add more robustness to the original Kalman Filter as well as removing the "need to know" a priori statistics about system noise. [6] This type of observer is known as the worst case observer since certain aspects of the system may not be known, thus maximizing the cost function. In plain terms this means the observer is more robust against un-modeled terms. The dynamics and updated equations of the H_∞ look similar to that of an EKF but the H_∞ filter has a few extra terms. It is possible to reduce the H_∞ Filter to a Kalman Filter based on the parameters chosen for the system. This method has been determined to be the most efficient way to create a robust Kalman-type Observer. [6]

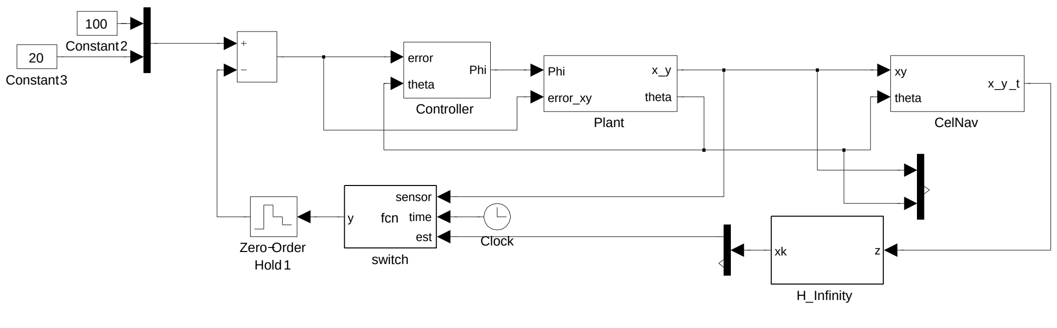

Figure 7.9: H-Infinity (H_∞) Simulation Model

7.2.1 H$_\infty$ Governing Equations

The H$_\infty$ filter assumes a system of the following form [5]:

$$\begin{aligned} x_{k+1} &= F_k x_k + w_k \\ y_k &= H_k x_k + v_k \\ z_k &= L_k x_k \end{aligned} \tag{7.11}$$

Here, w_k and v_k are process and measurement noise terms, respectively. z_k is the state to be estimated. L_k is a user-defined matrix necessary to estimate x_k. If L_k is assumed to be an identity matrix (full state estimation), then an H$_\infty$ observer becomes a Kalman filter.

The H$_\infty$ estimator is a cost minimization solution governed by the following cost function:

$$J_1 = \frac{\Sigma_{k=0}^{N-1}||z_k - \hat{z}_k||^2_{S_k}}{||x_k - \hat{x}_k||^2_{P_0^{-1}} + \Sigma_{k=0}^{N-1}(||w_k||^2_{Q_k^{-1}} + ||v_k||^2_{R_k^{-1}})} \tag{7.12}$$

Here, S_k, P_0, Q_k, and R_k are user-defined matrices that are symmetric and positive definite. The estimation goal is to minimize $(z_k - \hat{z}_k)$ based upon our initial conditions x_0 and noise terms.

The additional term to the EKF that defines the H$_\infty$ filter is $\theta\bar{S}_k P_k$ where S_k can be defined as a symmetric positive definite matrix that is pre-defined by the user and is shown to affect the K_k filter gains such that

$$\bar{S}_k = L_k^T S_k L_k \tag{7.13}$$

Where θ is defined as a performance bound which seeks to minimize J such that $J < \frac{1}{\theta}$. The resulting H$_\infty$ filter is given as:

$$K_k = P_k[I - \theta\bar{S}_k P_k + H_k^T R_k^{-1} H_k P_k]^{-1} H_k^T R_k^{-1} \tag{7.14}$$

$$\hat{x}_{k+1} = F_k\hat{x}_k + F_kK_k(y_k - H_k\hat{x}_k) \tag{7.15}$$

$$P_{k+1} = F_kP - K[I - \theta\bar{S}_kP_k + H_k^TR_k^{-1}H_kP_k]^{-1}F_k^T + Q_k \tag{7.16}$$

If $\theta = 0$, then the H_∞ becomes identical to that of the Extended Kalman Filter. [6]

The function below ensures that there is a valid estimator solution at each time step k:

$$P_k^{-1} - \theta\bar{S}_k + H_k^TR_k^{-1}H_k > 0 \tag{7.17}$$

If at any time-step "k" this function does not prove true, then the estimate for that time step is not a valid solution to the problem. For this above function to hold true and be positive-definite, the requirements on $\theta\bar{S}_k$ are such that:

- $\theta\bar{S}_k$ is small if θ is small.
- $\theta\bar{S}_k$ is small if L_k is small.
- $\theta\bar{S}_k$ is small if S_k is small.

7.2.2 H_∞ Simulation Results

Results from the closed-loop H_∞ as applied to the nonlinear simulated system with "expected" noise of $1 * 10^{-4}$ magnitude are shown in Figure 7.10 and Figure 7.11. The first trajectory graph is similar to the one presented in the EKF, but it can be seen that there is slightly more wobble in the trajectory plot (more clearly seen in Figure 7.11). This again is most likely do to the difficulty of this "rover" model to travel in a straight line. The rover does arrive at the desired location again without any arching paths, which means the rover did not deviate more in either the x or y directions. Below are the covariance matrices for use with both the "expected" and "worst case" noise scenarios:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad R = \begin{bmatrix} 2400 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2400 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

$$L = \begin{bmatrix} .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{7.18}$$

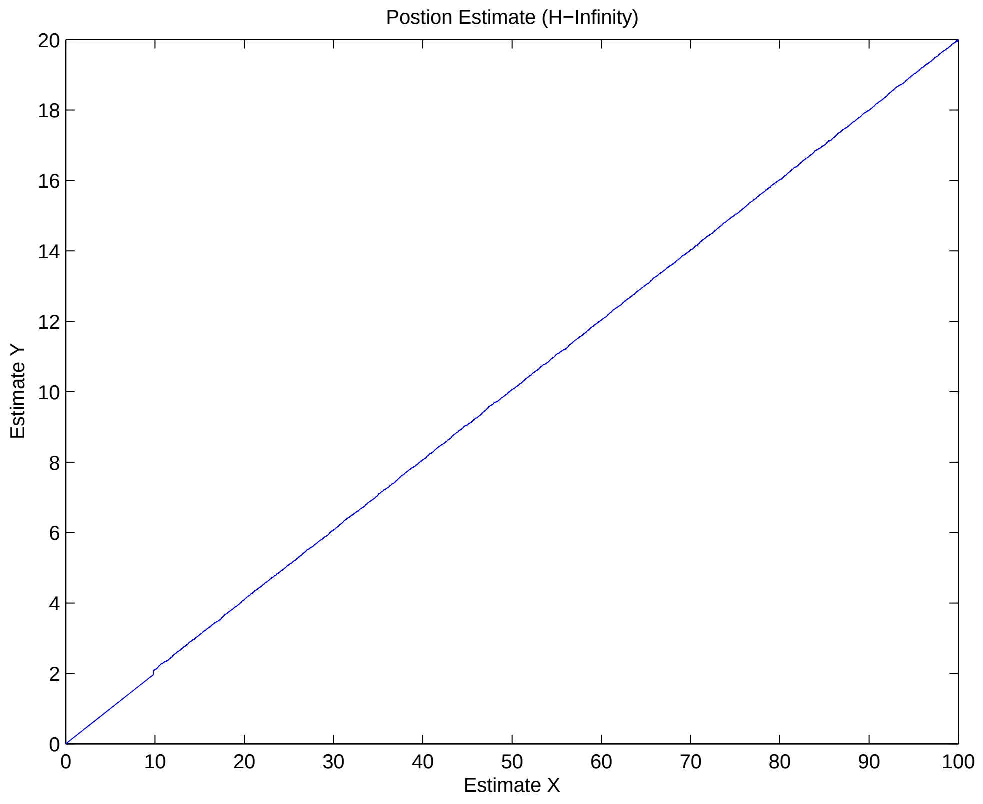

Figure 7.10: H_∞ - X and Y Estimate Trajectory Expected Noise

The position error (in degrees) is shown in Figure 7.11. Here, it can be seen that there is an offset only in the y direction, most likely due to the x direction having less estimation lag than the y direction. In Table 7.8 the mean of the position error is 0.0247 degrees in latitude and 0.1070 in longitude. Also important to note is the tight error band for Figure 7.11. Again the "rover" does not deviate significantly from the target point but does stray more than with the EKF with the same initial conditions.

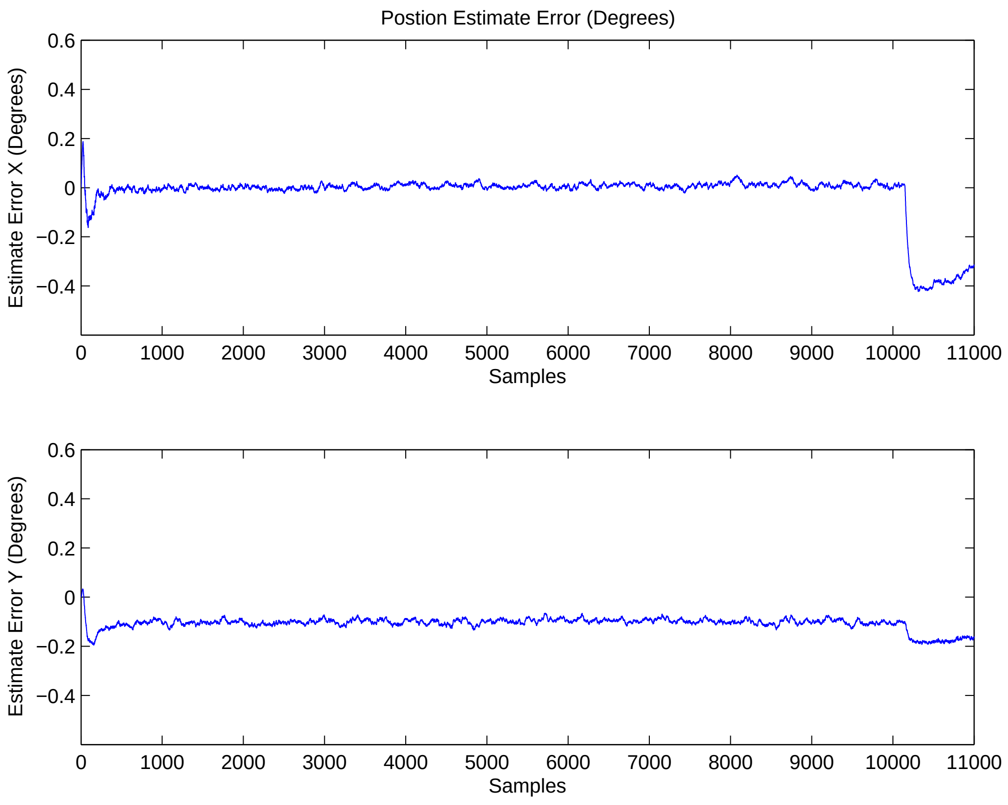

Figure 7.11: H_∞ - X and Y Estimate Error with Expected Noise (Degrees)

Mean [μ](Degrees)		Standard Deviation[σ](Degrees)	
Latitude	Longitude	Latitude	Longitude
0.0247	0.1070	0.1019	0.0251

Table 7.8: H_∞ - Mean [μ] and Standard Deviation [σ] for Higher Noise

As can be seen in Figure 7.12, the navigational error is approximately ±500 meters. This is to be expected due to the fact that EKF and H_∞ are so similar in application and the noise magnitude is held constant. Note that there are many more shape spikes in Figure 7.12 when compared to Figure 7.5. This is most likely caused by the choices in the covariance matrices potentially not being optimal. These sharp peaks can potentially be resolved by investing more time into choosing a better weighted covariance matrix. The mean and standard deviation for the navigational error can be found in Table 7.9. They are 1302.98m and 329.71m, respectively, in the x and y directions These values confirm the visual observation in Figure 7.12 of a ±500 meter error band. Less than ±500 meters error is considered a very good result and shows promise for greatly reducing navigational error with tuned observers.

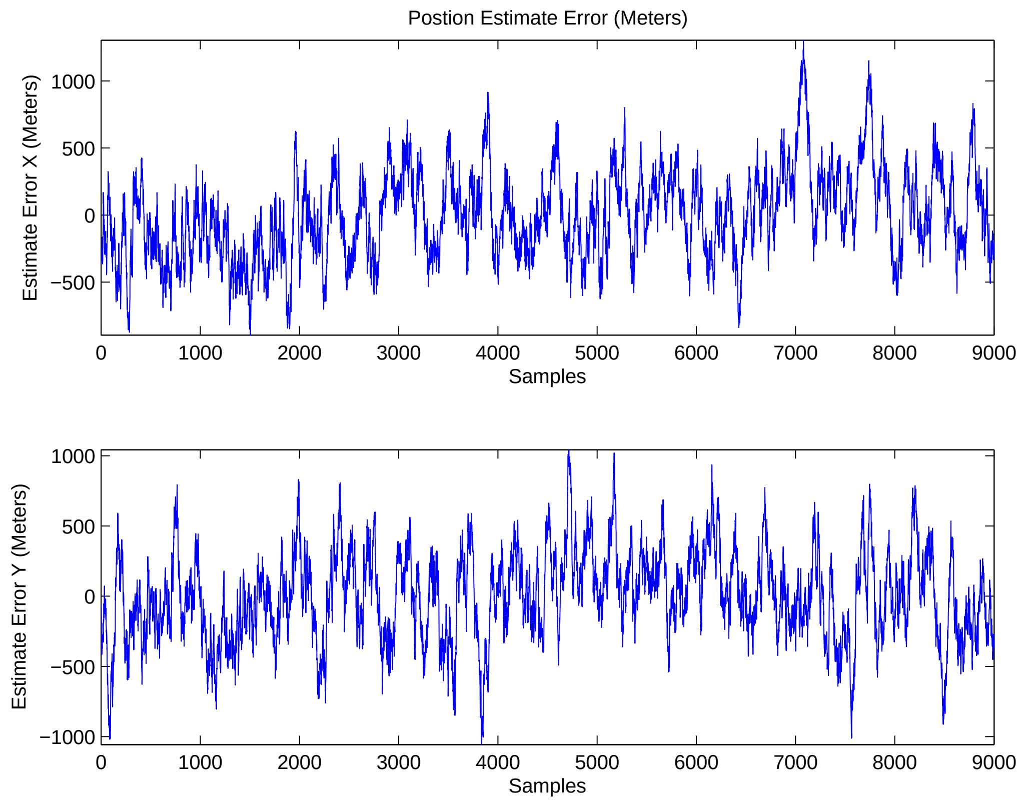

Figure 7.12: H_∞ - X and Y Estimate Error Expected Noise (Meters)

Maximum Navigational Error (Meters)		Standard Deviation [σ](Meters)	
Latitude	Longitude	Latitude	Longitude
1302.97	1043.39	329.716	316.914

Table 7.9: H_∞ - Maximum Navigational Error and Standard Deviation [σ] for Expected Noise

Results from the H_∞ as applied to the nonlinear simulated system with "worst" noise of $1 * 10^{-2}$ magnitude are seen in Figure 7.14, and to a lesser extent Figure 7.13. Again the first figure to be examined is the rover trajectory seen in Figure 7.13. There is a significant amount of "wobble" in the trajectory of the rover. The rover's movements are jagged sharp turns, which expends significant control effort. Also, as the "rover" begins to reach its desired location, the "rover" overshoots the location multiple times. This may be do to the fixed velocity condition of the "rover", such that if a variable speed is used the overshoot may be negated. In other words if the rover is allowed to decrease velocity as it approaches the desired location, this would reduce the chance of overshoot.

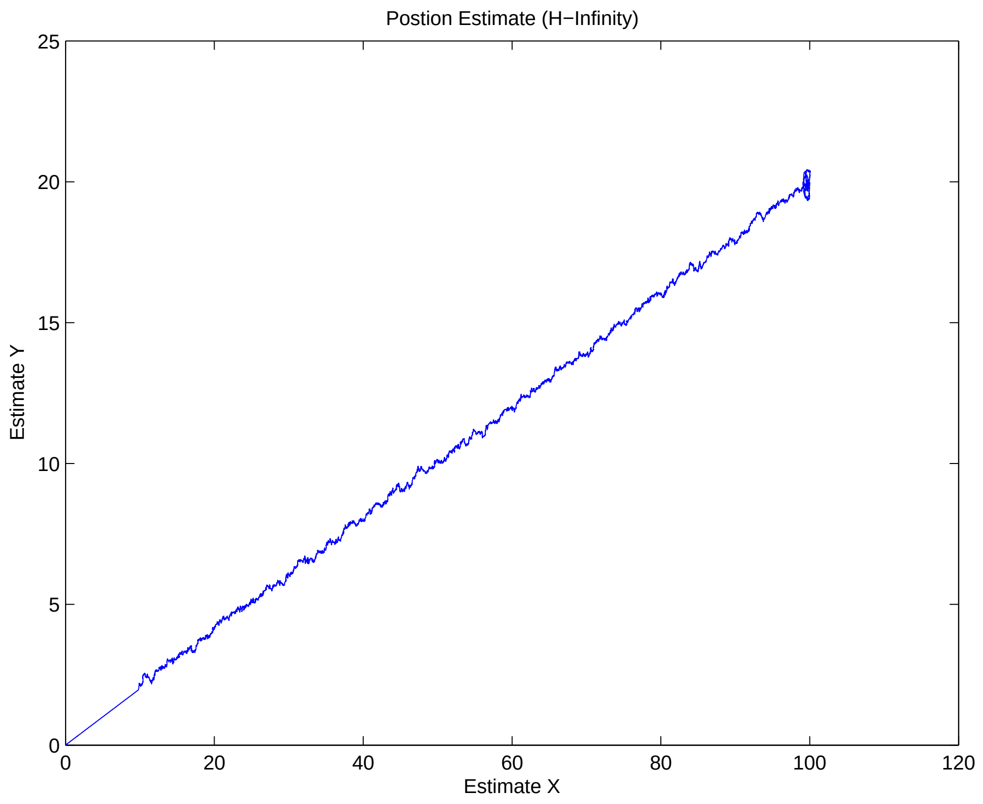

Figure 7.13: H_∞ - X and Y Estimate Trajectory with "Worst Noise" (Meters)

The H_∞ position error (in degrees) is shown in Figure 7.14. Here it can be seen again that there is an offset only in the y direction. In Table 7.10 the mean of the position error is 0.0123 degrees in latitude and 0.1045 degrees in longitude. The error band for Figure 7.14 is significantly larger than the expected noise case. This is consistent with the trajectory shown in Figure 7.14.

Mean [μ] (Degrees)		Standard Deviation [σ] (Degrees)	
Latitude	Longitude	Latitude	Longitude
0.0123	0.1045	0.1477	0.1101

Table 7.10: H_∞ Mean [μ] and Standard Deviation [σ] with "Worst" Noise (Degrees)

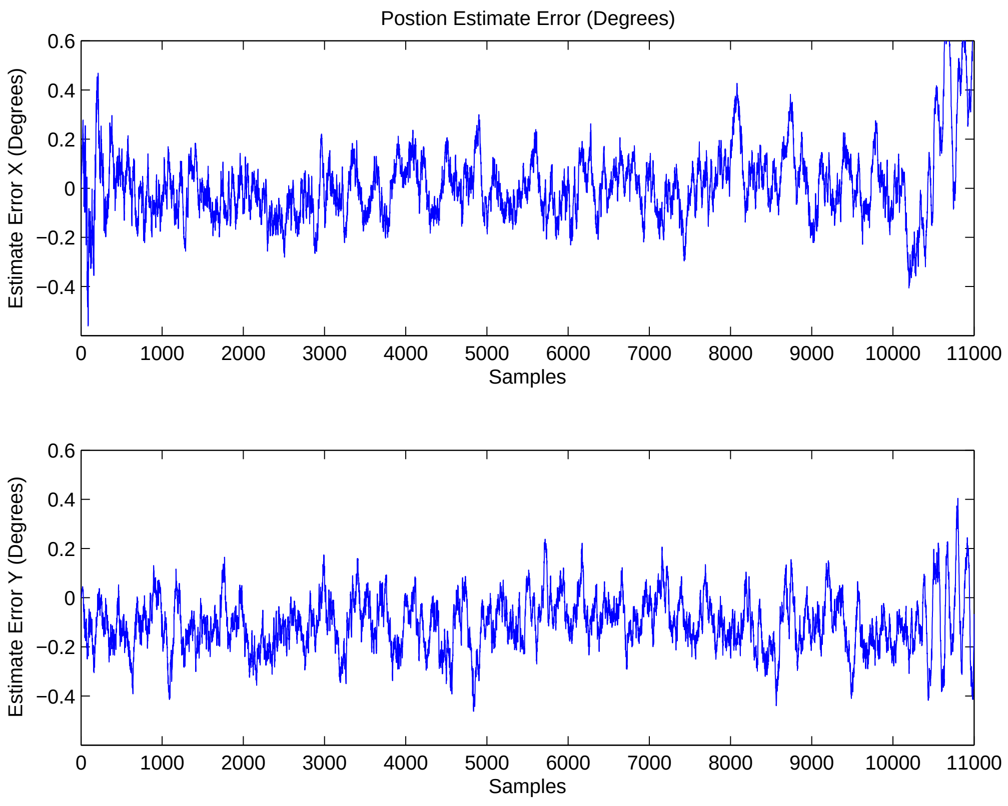

Figure 7.14: H_∞ - X and Y Estimate Error “Worst” Noise (Degrees)

In Figure 7.15 the navigational error is approximately ±5000 meters, similar to the navigational error seen in the EKF “worst” noise Figure 7.8, which is expected as the EKF and H_∞ filer are so closely related. Many sharp peaks can again be seen in Figure 7.15, similar to the expected noise simulation Figure 7.12. These can most likely be resolved by investing more time into tuning the PID controller. The mean and standard deviation for the navigational error, found Table 7.9, correspond to the visual information seen in Figure 7.15 and Table 7.11, with the maximum navigational error of 12494.29m and the standard deviation of 3147.54m. Again note that these values are outside the acceptable upper visual limit of 2000 meters.

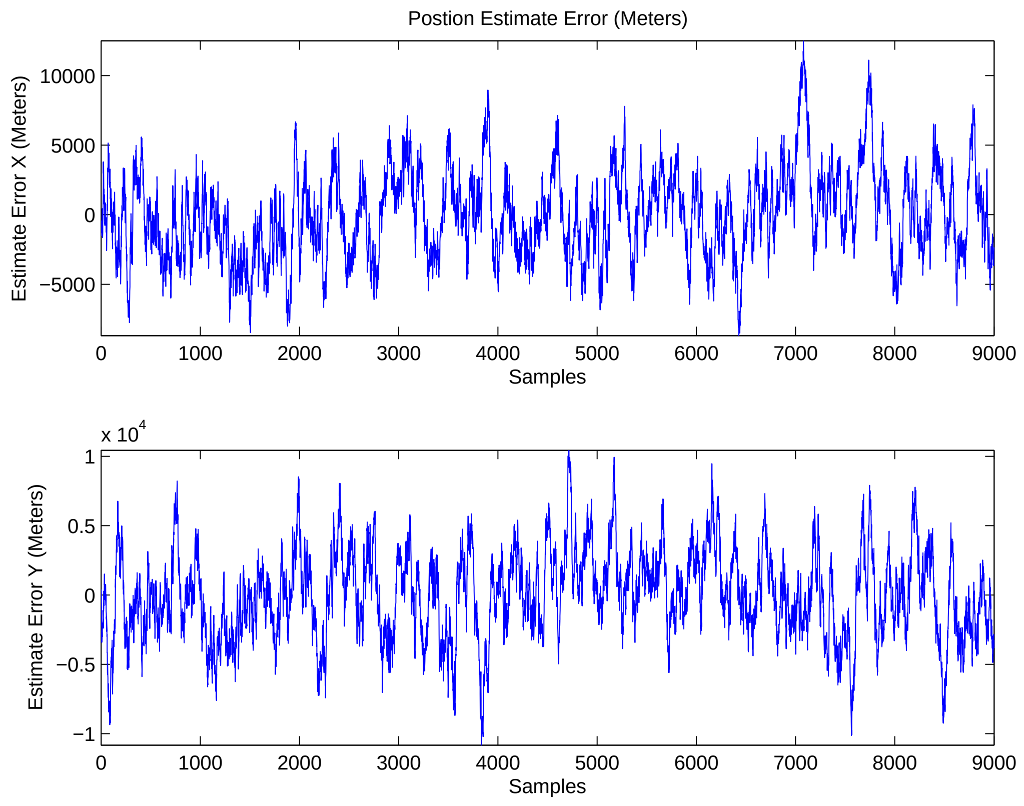

Figure 7.15: H_∞ - X and Y Estimate Error with "Worst" Noise (Meters)

Maximum Navigational Error (Meters)		Standard Deviation $[\sigma]$ (Meters)	
Latitude	Longitude	Latitude	Longitude
12494.29	10437.39	3147.54	3138.20

Table 7.11: H_∞ Maximum Navigational Error and Standard Deviation $[\sigma]$ with "Worst" Noise (Meters)

As seen with the EKF, there is also an offset in the H_∞ estimator for the "expected" case (Table7.5 and Table 7.9). Again, the remaining signal deviation is ± 0.02 degrees, in the x-direction. The same can be seen in the y-direction. The offset is seen again for the "worst" case, in both the x and y-direction. The remaining signal deviation is ± 0.2 degrees. The same can be seen in the y.

7.3 Sliding Mode Observer

The Sliding Mode Observer (SMO) as used in this thesis works on the assumption that it is easier to work with a first order system than to work with an n^{th}-order as defined by n^{th}-order differential equations [24]. SMO has the capability of providing excellent stability and performance when dealing with possible modeling errors.

SMO works in a two phase process: the reaching phase and the sliding phase. The reaching phase consists of the system error trajectory converging to a sliding surface(s) where the surface is defined such as s $= 0 = \dot{\tilde{x}} + \lambda\tilde{x}$. The sliding phase consists of the error trajectory moving along s. As long as the error trajectory remains on s, the error trajectory converges to the origin. Switching along the path of s is not instantaneous however, and chattering about the sliding surface tends to occur.

7.3.1 SMO Governing Equations

The SMO assumes the system of the following form:

$$\begin{aligned} x_{(k+1)} &= f_k(x_k) + B_k u_k + \eta_k \\ y_k &= g_k(x_k) + \nu_k \end{aligned} \tag{7.19}$$

One difference for the SMO, when compared to EKF and the H_∞, is the lack of a covariance matrix, such that the standard form of the SMO is:

$$\begin{aligned} \hat{x}_0 &= E(x_0) \\ \hat{x}_{k|k-1} &= f(\hat{x}_{k-1}) \\ \tilde{y} &= y_k - g(\hat{x}_{k|k-1}) \\ 1(S) &= sat(\alpha\tilde{y}) \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + G_{SMO}\tilde{y} - K_{SMO}1(S) \end{aligned} \tag{7.20}$$

For this system $\mathbf{1(S)}$ is defined as a switching function, which is in turn a function of the defined sliding surface **S**, Figure 7.16. This function is occasionally defined as either a signum or saturation function. The saturation function is defined below:

$$\begin{aligned} sat(s) &= +1 \quad \text{if } s > 0 \\ sat(s) &= -1 \quad \text{if } s < 0 \end{aligned} \tag{7.21}$$

The saturation function is favored in this research due to the increased chattering reduction. The SMO is much less computationally intensive than either EKF or H_∞ due to not needing to calculate a matrix inverse at each iteration. The disadvantage of SMO is that it has decreased resistance to measurement noise. [25]

7.3.2 SMO Simulation Results

Results from the closed-loop SMO as applied to the nonlinear simulated system with "expected" noise of $1 * 10^{-4}$ magnitude are shown in Figure 7.17 and Figure 7.18. It can be seen in Figure 7.17 that there is noticeable low level "wobble" on the trajectory and these levels are significantly higher than that of the EKF and H_∞ methods. This is most likely do to the position estimate oscillating

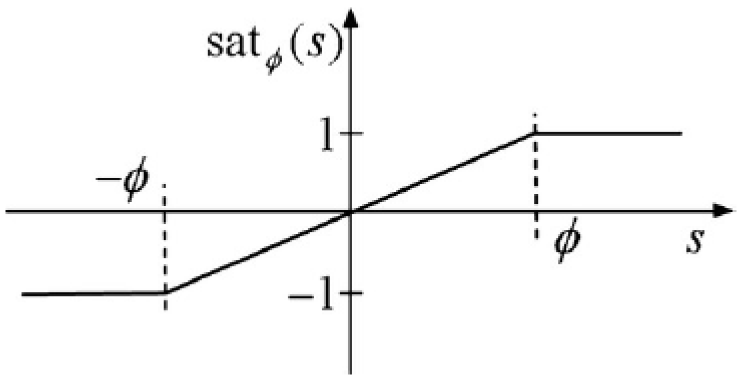

Figure 7.16: Sliding Mode Observer Saturation Function

along the sliding surface. The rover, however, is still able to travel directly to the desired location.

Below the gain choice for the SMO is shown: As seen previously in the

Phi		
x	y	θ
0.01	0.01	2

Table 7.12: Phi Gains

	Gains		
	x	y	θ
K	1	1	.1
C	1	1	1

Table 7.13: Gains

EKF Figure 7.4, there is again an offset in both the x and the y directions with the x being more significant than the y direction. It is possible to see this in greater detail in Table 7.14 with the mean of the x-direction being 0.0917 degrees and the y-direction being 0.0121 degrees, with the error band for both the x and y being approximately ± 0.04 degrees. When compared to the other two estimator, EKF

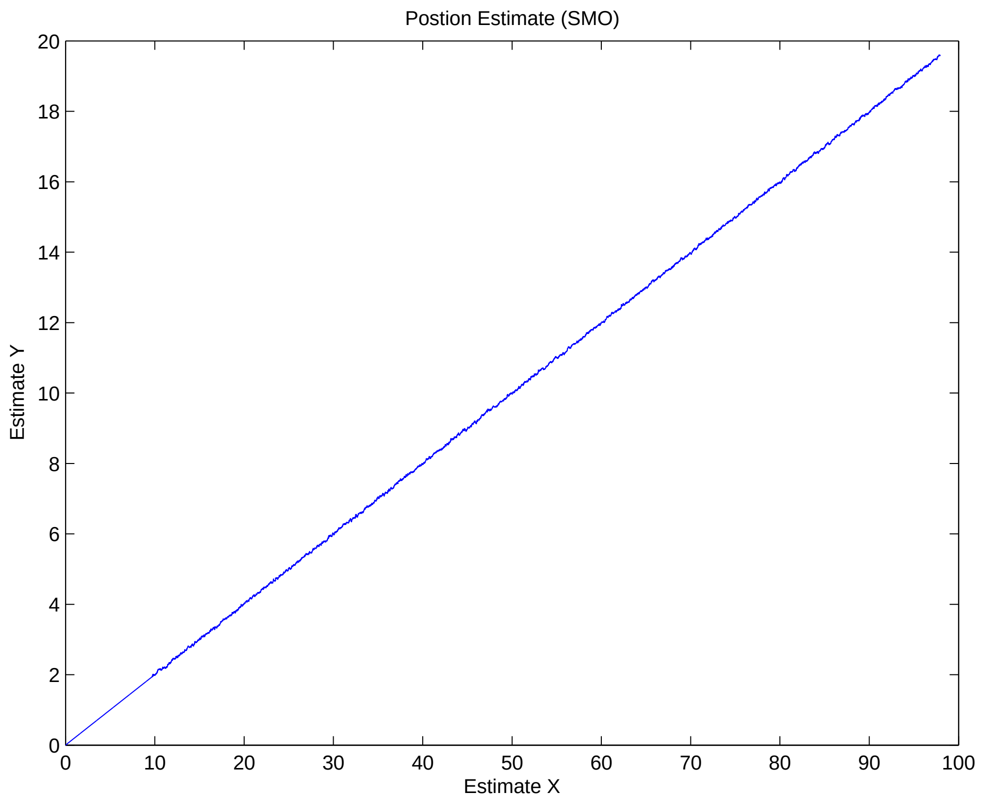

Figure 7.17: SMO - X and Y Estimate Trajectory with Expected Noise

and H_∞, the SMO has overall less position error.

Mean [μ] (Degrees)		Standard Deviation [σ] (Degrees)	
Latitude	Longitude	Latitude	Longitude
0.0917	0.0121	0.2199	0.2290

Table 7.14: SMO Mean [μ] and Standard Deviation [σ] for Expected Noise (Degrees)

In Figure 7.19 it can be seen that the navigational error is approximately $\pm$2000 meter, which this is at the upper limit of the acceptable vision range of 2000 meters. This means that some of the estimated positions measurements are outside of the acceptable visual boundary and would likely need to be ignored.

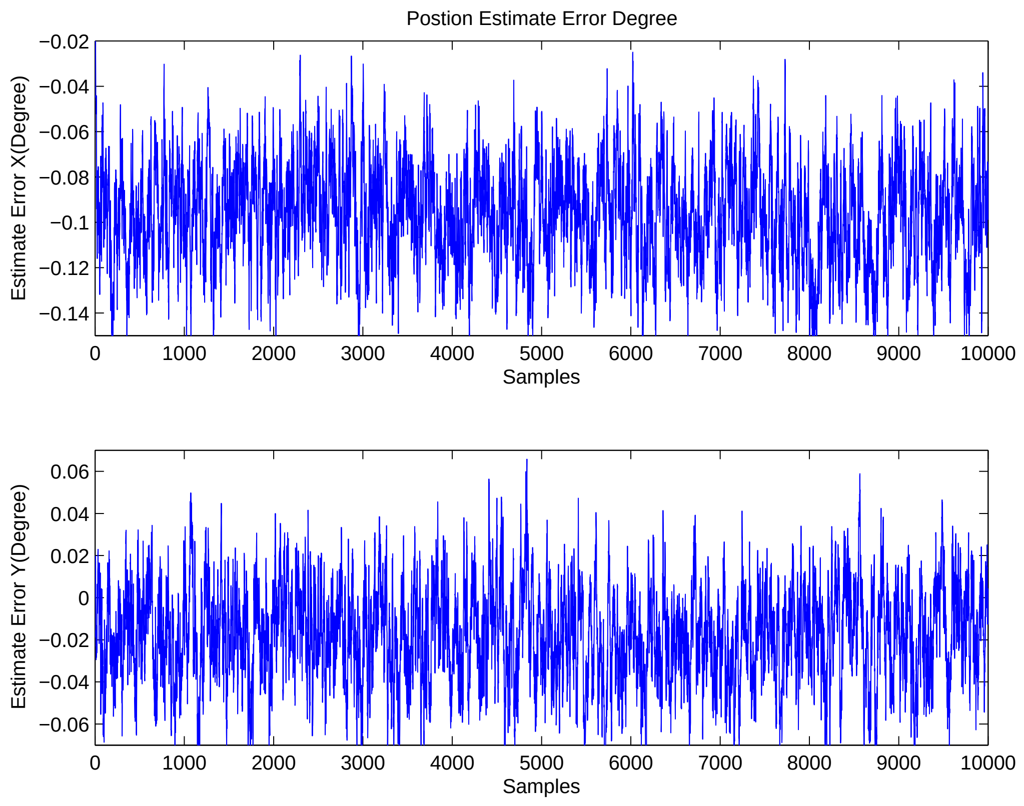

Figure 7.18: SMO - X and Y Estimate Error “Expected” Noise Levels (Degrees)

The large error is most likely due to overshoot when approaching the sliding surface and a difficultly staying on the sliding surface. It can be seen in Table 7.15 that the maximum navigational error is 2646.98m and the standard deviation is 702.18m. This is greater than both the EKF and H_∞ techniques.

Maximum Navigational Error (Meters)		Standard Deviation [σ] (Meters)	
Latitude	Longitude	Latitude	Longitude
2003.31	2646.98	652.87	702.18

Table 7.15: SMO Maximum Navigational Error and Standard Deviation [σ] with Expected Noise

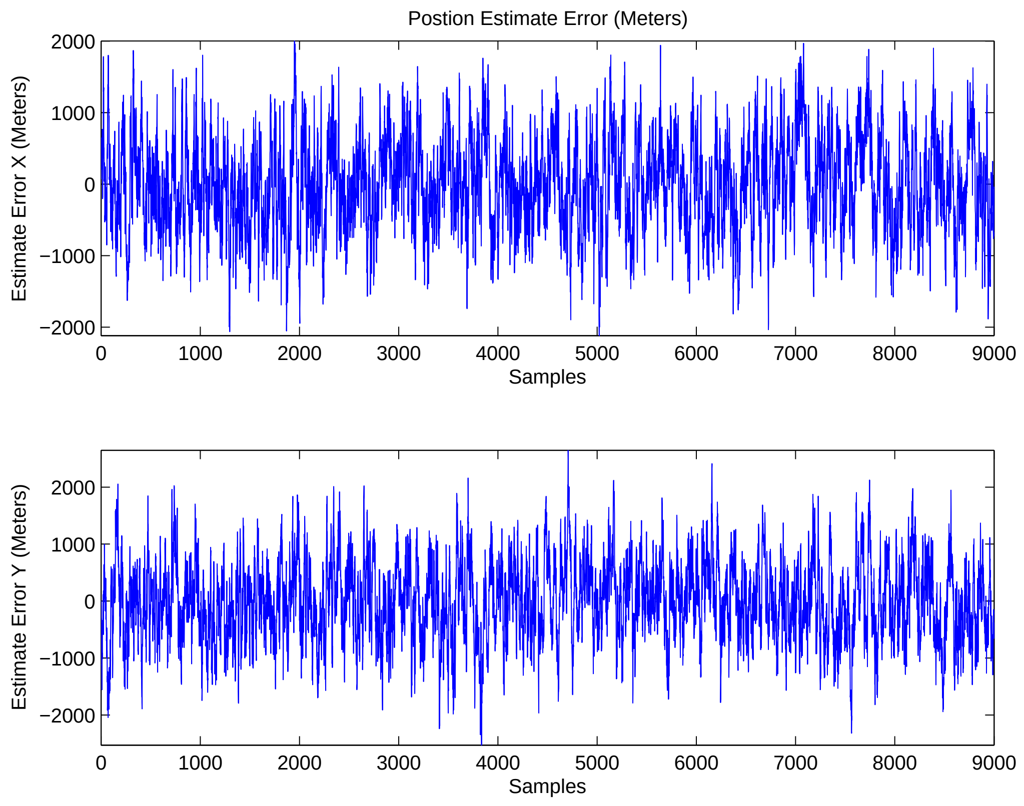

Figure 7.19: SMO - X and Y Estimate Error with "Expected" Noise Levels

Results from the closed-loop SMO as applied to the nonlinear simulated system with "worst" noise of $1 * 10^{-2}$ magnitude are shown in Figure 7.20 and Figure 7.21. It can be seen in Figure 7.20 that there are high levels of noise on the trajectory plot, and it easily has the highest noise band of all the estimators. This means that a significant amount of control effort is required to be expended in order for the "rover" to continue on its relatively straight path. It should be noted there is no overshoot, even with all the noise on the trajectory of the SMO, as compared to the H_∞, which showed a significant amount of overshoot. This is most likely do to the position estimate oscillating about the sliding surface.

Different from the previous "worst" case instances, there is little to no offset

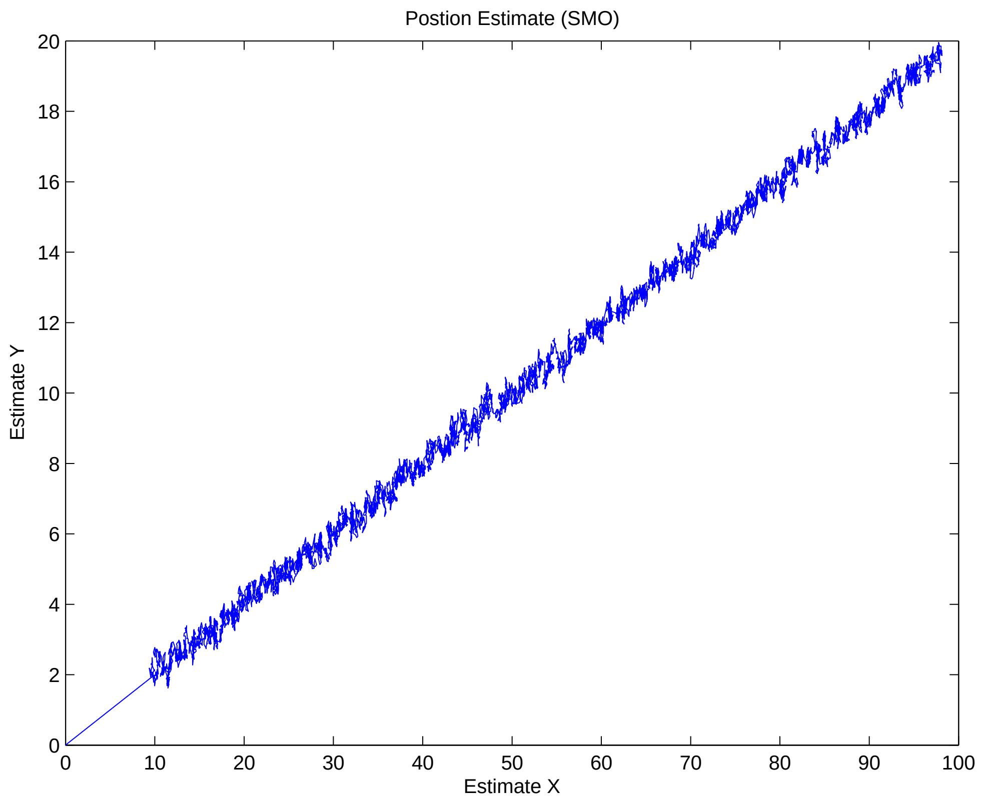

Figure 7.20: SMO - X and Y Estimate Trajectory with "Worst Case" Noise

in either the x and the y directions. This can be seen in Table 7.16 with the mean of the x-direction being 0.0935 degrees and the y-direction being 0.0175 degrees, with the error band for both the x and y about ± 0.6 degrees. A standard deviation of the position error in the x-direction, 0.0271 degrees, and the y-direction, 0.0229 degrees, can also be found in Table 7.16. Although there is high position error it should be noted that there is a very tight error band, as opposed to the previous estimators where error bands are more erratic.

Mean [μ] (Degrees)		Standard Deviation [σ] (Degrees)	
Latitude	Longitude	Latitude	Longitude
0.0935	0.0175	0.0271	0.0229

Table 7.16: SMO Mean [μ] and Standard Deviation [σ] for "Worst" Noise

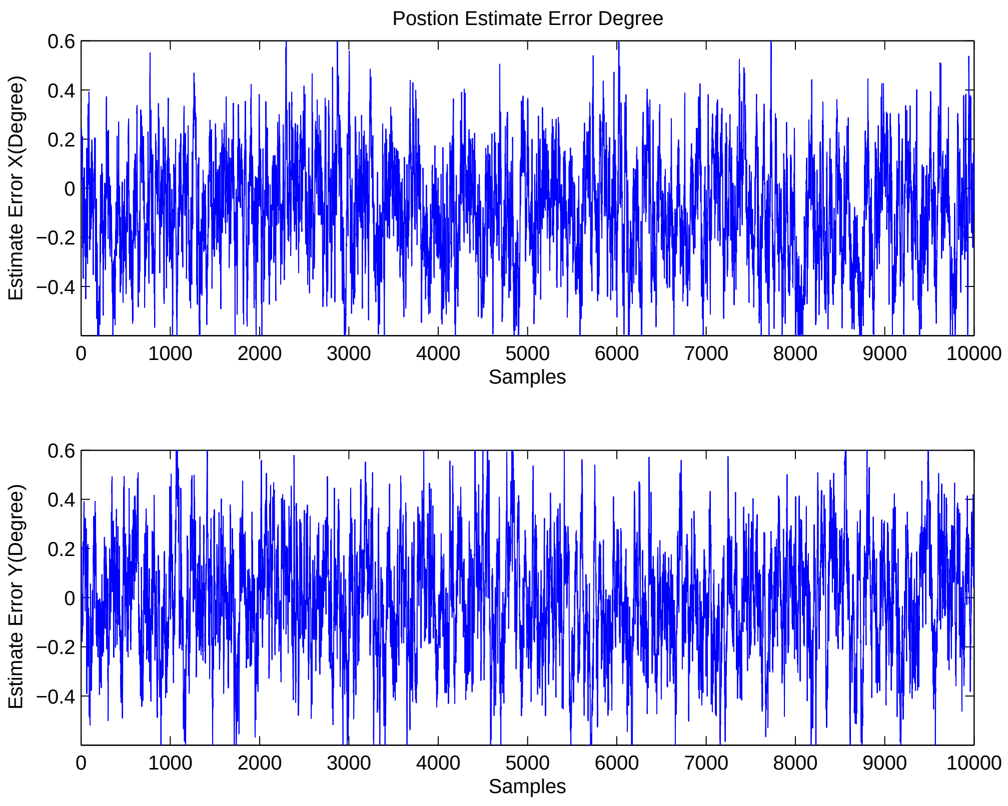

Figure 7.21: SMO - X and Y Estimate Error "Worst" Noise Levels

In Figure 7.22 it can be seen that the navigational error is approximately $\pm$20000 meters. This is the largest navigational error of all the estimators, being almost double the EKF or H_∞ for the equivalent error magnitude. The large error is due to excessive chattering along the sliding surface. This could be due to the signal band being too large so the estimation is not able to settle out. Table 7.17 shows that the maximum navigational error is 26787.03m and the standard deviation is 7013.25m. This is greater than the previous two estimators.

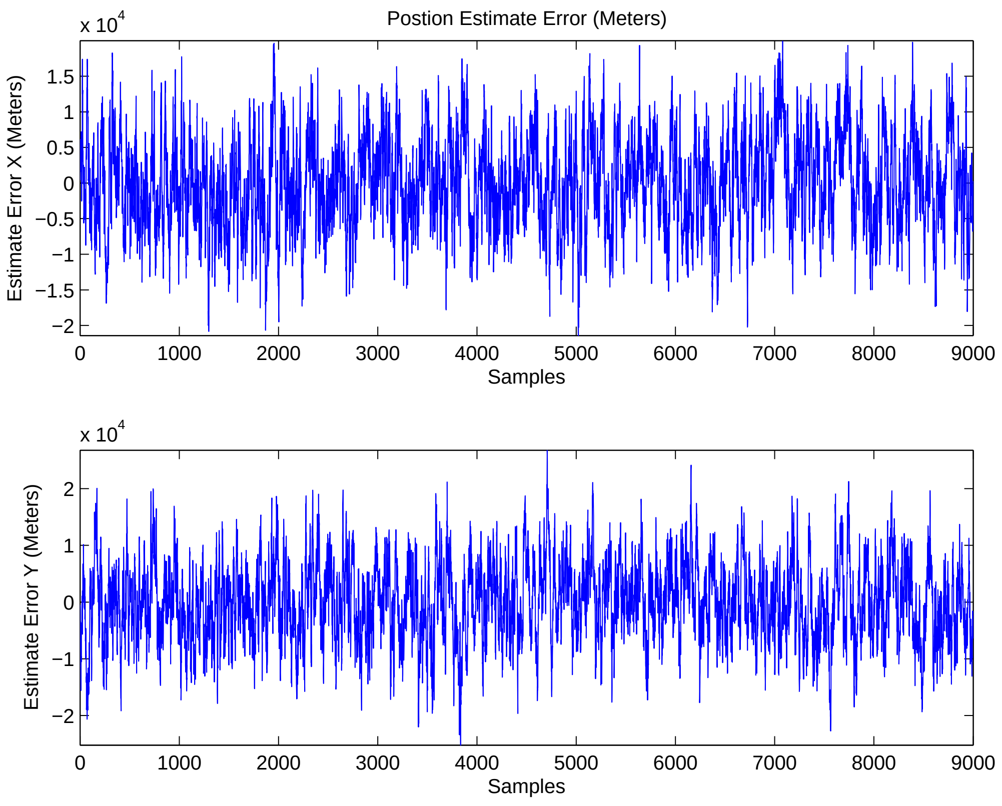

Figure 7.22: SMO - X and Y Estimate Error “Worst” Noise Levels in Meters

Maximum Navigational Error (Meters)		Standard Deviation [σ] (Meters)	
Latitude	Longitude	Latitude	Longitude
19971.82	26787.03	6543.59	7013.25

Table 7.17: SMO Maximum Navigational Error and Standard Deviation [σ] with “Worst” Noise (in Meters)

7.4 Comparative Analysis of Estimation Techniques

After examining each of these estimators individually, the EKF and H_∞ behave similarly, as expected. The SMO has the fastest simulation time. This is due to not having to perform matrix inverse operation at each time step. H_∞ has the best performance of the three estimators. All of the estimators are able to converge and all are able to reach the desired location. It must almost be noted that both EKF and H_∞ did have an offset error because the system lags behind by two time steps. This, however, does not prevent the filters' estimates from converging.

Chapter VIII

Conclusions

This research reviews topics regarding CelNav.

1. Statistical Analysis
2. SkyScout Simulations
3. Analytical Simulations

The statistical analysis shows CelNav reacting extremely well at different levels of noise and at different latitudes and longitudes. This can be seen in Table 4.3. This can further be seen when comparing Figure 4.12 and Figure 4.4, where even with high levels either at the North Pole or the Equator the maximum error is 2000 meters, which again is at the upper end of the viewable area on the Lunar surface. CelNav can still accurately determine latitude and longitude. The tables and figures, shown in Chapter IV, are a good preliminary indication of the versatility of the CelNav algorithm.

As shown in the Celestron experiment even small angular errors, as little as 0.001 degrees, can translate into large navigational errors on the scale of hundreds to thousands of meters, as seen in Table 5.8. Both trials are good representations of using an experimental sensor in conjunction with CelNav, where the error for both trials are very close to each other. The Celestron Skyscout is an experimental sensor platform for preliminary real world experimental testing, although repeatability is difficult due to standard GPS error. In the future the Celestron Skyscout could and should be integrated into a test platform as an accurate sensor platform for comparison to the proposed experimental setup.

The length of time to perform analytical experiments show insight into the computational power and memory that are required for both CelNav and the proposed estimators (EKF, H_∞ and SMO). This information is important for the future work of determining an effective controller that will also be integrated into the simulation. These results could be further improved upon with the addition of a tuned control system.

It was found that the EKF and H_∞ have very similar results, with the H_∞ having the better results, when comparing maximum navigational error (EKF - 14000m,H_∞ - 12000m, and SMO - 26000m). Although the H_∞ has the best results the SMO is computationally the fastest of the three estimators. Additionally all three estimators converge, showing stability in the system. H_∞ also had the least amount of “wobble” and SMO the most when comparing Figure 7.6, Figure 7.13, and Figure 7.20. With moderate noise of 10^{-6} all of the estimators results' are well within the 2000m upper visual limit of sight on the Lunar surface. These simulations show that the H_∞ has the fastest response time and the least residual noise.

Future analysis of CelNav-based state estimate involves repeating the statistical analysis with the following combinations: (1) CelNav and estimator, (2) CelNav and controller and (3) CelNav estimator and controller. Future experimental testing involves an experimental validation of CelNav-based navigation using various terrain and geographical locations.

The goal of this research it to further the development of an extraterrestrial mobility solution. Future research should incorporate an observer-based control for autonomous navigation. Testing would involve a comparison of CelNav-based navigation determination to GPS readings. Future testing should incorporate observer-based controllers to determine a low-memory, low-power autonomous mobility solution, with a focus on minimizing computational effort and control effort, while increasing robustness to noise, CelNav error, and external disturbances.

Bibliography

[1] Swanzy, Michael J. and Mortari, Daniele and Hurtado, John E. and Junkins, John L., "Analysis and Demonstration: A Proof-of-Concept Compass Star Tracker", *Proceedings of the 2006 AAS/AIAA Spaceflight Mechanics Meeting*, AAS 06-145, 2006.

[2] Thein, May-Win L., "Celestial Navigation (CelNav): Lunar Surface Navigation," AIAA-2008-6759, *Proceedings of the 2008 AIAA/AAS Astrodynamics Specialist Congress and Exposition*, Honolulu, HI, August 2008.

[3] Kalos, Marvin H., Whitlock, Paula A.
"Monte Carlo Methods."
Wiley-Blackwell, New York, 2008.

[4] Rubinstein, Reuven Y., Kroese, Dirk P.
"Simulation and the Monte Carlo Method."
John Wiley and Sons,New Jersey, 2008.

[5] "Time and Frequency" National Institute of Standards and Technology October 5. 2010 [http://www.nist.gov/pml/div688/grp40/enc-s.cfm. Accessed 6/27/12]

[6] Simon, Dan,

Optimal State Estimation: Kalman, H-infinity, and Nonlinear Approaches,

John Wiley and Sons, New Jersey, 2006.

[7] Welch, Greg, Bishop, Gary.

"An Introduction to the Kalman Filter"

UNC-Chapel Hill, TR 95-041 July 24 2006.

[8] Sun, Tsung-Ying.

"Particle Swarm Optimizer base Controller Design for Vehicle Navigation Systems"

International Conference on Systems, Man and Cybernetics IEEE 2008.

[9] Hoag, David.

"Apollo Guidance and Navigation Considerations of Apollo IMU Gimbal Lock"

April 1963

`http://www.hq.nasa.gov/alsj/e-1344.htm`

[10] Curtin University.

"Planetary and Lunar Gravity Field Modelling"

February 23 2012 `http://geodesy.curtin.edu.au/research/models/`

[11] Ifland, Peter Commander in the US Naval Reserve

"The History of the Sextant"

October 3, 2000

`http://www.mat.uc.pt/~helios/Mestre/Novemb00/H61iflan.htm`

[12] National Academy of Sciences

"GPS: The Role of Atomic Clocks"

2009

http://www.beyonddiscovery.org/content/view.page.asp?I=464

[13] The United Kingdom Hydrographic Office & U.S. Naval Observatory

"2012 U.S. Naval Observatory Nautical Almanac"

Skyhorse Publishing, New York, NY, January 26, 2012

[14] Corum, C. E., NAVAL RESEARCH LAB WASHINGTON DC

"Navigation by Moon Doppler Effect"

20 JAN 1959

[15] Benjamin P. Malay,

"CELESTIAL NAVIGATION ON THE SURFACE OF MARS"

05 July 2001 US Naval Academy Annapolis, MD

http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2\&doc=GetTRDoc.pdf\&AD=ADA392455

[16] R. Trautner et. al.

"A NEW CELESTIAL NAVIGATION METHOD FOR MARS LANDERS"

2004 Lunar and Planetary Science XXXV

http://www.lpi.usra.edu/meetings/lpsc2004/pdf/1106.pdf

[17] Winkler, Gernot M. R.

"MODIFIED JULIAN DATE"

November 20, 2002 http://tycho.usno.navy.mil/mjd.html

[18] U.S. Navel Observatory

"Approximate Sidereal Time"

August 29, 2012

`http://www.usno.navy.mil/USNO/astronomical-applications/`

`astronomical-information-center/approx-sider-time`

[19] Universal Serial Bus (UBS)

"USB 3.0 Specification"

April 28, 2013

`http://www.usb.org/developers/docs/`

[20] Gaspar,Joaquim Alves

"Marine Sextant"

October 2006

`http://en.wikipedia.org/wiki/File:Marine\_sextant.svg`

[21] Brown, Hart

"What is GPS?"

Monday, May 13, 2013

`http://www.mooreschools.com/page/19655`

[22] Quinn, David

NASA Aerospace Engineer

Summer 2007

[23] Gelb, Arthur

"Applied Optimal Estimation"

The MIT Press 1974

[24] Slotine, Jean-Jacques

“Applies Nonliner Control”

Prentice-Hall Inc 2004

[25] Jenkins, Ben

“Accelerometer Calibration for NASA’s Magnetospheric Multiscale Mission Spacecraft”

Fall 2010 University of New Hampshire

[26] Koprubasi, Kerem

“Spacecraft Attitude and Angular Rate Estimation Using Sliding Mode Observers”

Fall 2003 University of New Hampshire

[27] Malan, DF

“Euler Angles, General Convention”

September, 14 2004

`http://upload.wikimedia.org/wikipedia/commons/7/73/EulerG.png`

[28] Mahoney, MJ

“Pointing an Instrument on an Airborne Platform”

Friday, October 19, 2007

`http:`

`//mtp.mjmahoney.net/www/notes/pointing/Aircraft\_Attitude2.png`

[29] Lucas

“Showing 4-axis aligned, still exhibiting gimbal lock problem”

February, 12 2011

http://upload.wikimedia.org/wikipedia/commons/7/7d/Gimbal_lock_still_occurs_with_4_axis.png

[30] Williamson, David

"The Evolution of Dynamical Cores for Global Atmospheric Models"

October, 26 2007

http://royalsocietypublishing.org/content/roypta/367/1890/833/F4.large.jpg

[31] Underwood, Amy

"Celestial Navigation-Based Controller Design for Extra-Terrestrial Surface Navigation"

Fall 2013 University of New Hampshire

[32] Henderson, D.M

"Euler Angles, Quaternions, and Tranformation Matrices"

July 1977

http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770024290.pdf

Summary accessed from:http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles

APPENDICES

Appendix A

CelNav Algorithms

A.1 Original CelNav Algorithm

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

%   run_data_xxxx.m - extension of data_gen_xxxx.m
%                   confirms calculations of lambda, phi
%
%         - calculates Gamma ("truth" data) given Delta and SC
%             - yields: unit vector Gamma
%                     unit quaternion Delta
%             - confirms normalization of quaternions
%         - recalculates lambda, phi, and epsilon given Gamma and Delta
%         - calls rand_q.m, xprod_mat.m, qprod.m, q2rotmat.m
%
%         - generates vectors of data
%         - test for lander "sight" viewing angle (Beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

%   Variable Definitions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% R_x = rotation matrix for x
% x_vec = vector for x
%
% eta = unit vector (defined for quaternion)
% thetha = rotation angle (defined for quaternion)
%
% Gamma --> Accelerometer
% Delta --> Star Tracker
% Phi  --> M__ENU_SD (function of lambda and phi)
% Omega --> Moon
%
% n = number of star tracker measurements
% n_lambda = iterations of lambda (full 0 to +360 degree coverage)
% n_phi = iterations of phi (full -90 to +90 degree coverage)
% d - 1 = number of acceptable data points
%
% Beta = lander "sight" angle (not boresight angle) according to 30 deg
%         crater slope
%
% "lambda" = longitude
% "phi" = lattitude
%
% lambda_true = lambda truth data
% phi_true = phi truth data
% lambda_mR = extracted lambda
% phi_mR = extracted phi
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear
close all;

%N = 1;                      % Number of tests generated
Beta = 60;
Body_z = [0 0 1]';

%j=1;
n=1;
% n_lambda=36;
% n_phi=37;
% n_epsilon=36;
n_lambda=10;
n_phi=10;
n_epsilon=6;
d=1;
moon_long = 1;
moon_lat = 1;
count(1,1) = 0;
i = 1;
e = 1;
max_lambda=0;
min_lambda=0;
max_phi=0;
min_phi=0;
rad_fac = 1e0;

% Define Accelerometer alignment
% mag_noise_Gamma = 5e-5;
% mag_noise_Gamma = 1e-6;
mag_noise_Gamma = 1e-6;
Acc_vec = [0 0 1]';          % Accelerometer z-vector
Acc_down = -Acc_vec;
Gamma__Body_sAcc_eta = cross(Acc_vec,Body_z);
Gamma__Body_sAcc_theta = acos(Acc_vec'*(Body_z)); % degrees
Gamma__Body_sAcc_quat = Gamma__Body_sAcc_eta*sin(Gamma__Body_sAcc_theta
    /2);
Gamma__Body_sAcc_quat0 = cos(Gamma__Body_sAcc_theta/2);
R_Gamma__Body_sAcc = q2rotmat(Gamma__Body_sAcc_quat,
    Gamma__Body_sAcc_quat0);

% Define Star tracker alignment
% mag_noise_Delta_theta = 1e-3;
% mag_noise_Delta_theta = 3e-3;
mag_noise_Delta_theta = 3e-6;
Star_tracker_vec = [0 0 1]';  % Star tracker z-vector
Delta__Body_sST_eta = cross(Star_tracker_vec,Body_z);
Delta__Body_sST_theta = acos(Star_tracker_vec'*Body_z); % degrees
Delta__Body_sST_quat = Delta__Body_sST_eta*sin(Delta__Body_sST_theta/2);
Delta__Body_sST_quat0 = cos(Delta__Body_sST_theta/2);
R_Delta__Body_sST = q2rotmat(Delta__Body_sST_quat,Delta__Body_sST_quat0)
    ;

% Star_tracker_plane = 2;      % 1=yz plane, 2=xz plane, 3=xy plane
                               % 4=xyz plane
% if Star_tracker_plane = 1,
%    rotate_Star_tracker = ;
% else
% end;

% Define lunar coordinate transformation
R_moon = 1738.2*1e3;  % m
Omega__SC_I = eye(3);
Omega__SD_SC = eye(3);
Omega__SD_I = Omega__SD_SC*Omega__SC_I;
R_Omega__SD_I = Omega__SD_I;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate random star tracker quaternion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[Delta_quat,Delta_quat0] = rand_q(n);

R_Delta_raw = q2rotmat(Delta_quat,Delta_quat0);

% for p=1:n,
%    Delta_vec(:,p) = R_Delta_raw(:,:,p)*Star_tracker_vec;
% end;

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

```
%   Lunar Surface Map
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Latitude lines
for moon_lat_p=1:5,
    phi_lat = (moon_lat_p-1)*5 - 90;
    for moon_lat_l=1:19,
        lam_lat = (moon_lat_l-1)*20;
        Moon_lat_z(moon_lat) = R_moon*sin(phi_lat*pi/180);
        Moon_lat_x(moon_lat) = R_moon*cos(lam_lat*pi/180)*cos(phi_lat*pi
            /180);
        Moon_lat_y(moon_lat) = R_moon*sin(lam_lat*pi/180)*cos(phi_lat*pi
            /180);
        moon_lat=moon_lat+1;
    end;
end;

% Longitude lines
for moon_long_l=1:19,
    lam_long = (moon_long_l-1)*20;
    for moon_long_p=1:5,
        phi_long = (moon_long_p-1)*5 - 90;
        Moon_long_z(moon_long) = R_moon*sin(phi_long*pi/180);
        Moon_long_x(moon_long) = R_moon*cos(lam_long*pi/180)*cos(phi_long
            *pi/180);
        Moon_long_y(moon_long) = R_moon*sin(lam_long*pi/180)*cos(phi_long
            *pi/180);
        moon_long=moon_long+1;
    end;
end;

moon_long = 1;
moon_lat = 1;

% Latitude lines
for moon_lat_p=1:19,
    phi_lat = (moon_lat_p-1)*10-90;
    for moon_lat_l=1:19,
        lam_lat = (moon_lat_l-1)*20;
        Moon_Lat_z(moon_lat) = R_moon*sin(phi_lat*pi/180);
        Moon_Lat_x(moon_lat) = R_moon*cos(lam_lat*pi/180)*cos(phi_lat*pi
            /180);
        Moon_Lat_y(moon_lat) = R_moon*sin(lam_lat*pi/180)*cos(phi_lat*pi
            /180);
        moon_lat=moon_lat+1;
    end;
end;

% Longitude lines
for moon_long_l=1:19,
    lam_long = (moon_long_l-1)*20;
    for moon_long_p=1:19,
        phi_long = (moon_long_p-1)*10-90;
        Moon_Long_z(moon_long) = R_moon*sin(phi_long*pi/180);
        Moon_Long_x(moon_long) = R_moon*cos(lam_long*pi/180)*cos(phi_long
            *pi/180);
        Moon_Long_y(moon_long) = R_moon*sin(lam_long*pi/180)*cos(phi_long
            *pi/180);
        moon_long=moon_long+1;
    end;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Generate test data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for j=1:n,

    old_d = d;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Define Star Tracker alignment and measurement errors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  Define Star Tracker Measurement Errors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Generate random noise vector eta and angle theta
%     noise_Delta_vec = mag_noise_Delta*(2*rand(3,1) - 1); % uniform
    noise
    noise_Delta_vec =(rand(3,1) - 0.5); % uniform noise
%     noise_Delta_quat = mag_noise_Delta*(2*randn(3,1) - 1); % normaly
    distributed noise
    noise_Delta_eta = noise_Delta_vec/norm(noise_Delta_vec);
    noise_Delta_theta = mag_noise_Delta_theta*(2*randn - 1);
%     Delta_m = Delta_quat(:,j) + noise_Delta_quat;

    % Obtain normalized noise quaternion
    noise_Delta_q_raw = noise_Delta_eta*sin(noise_Delta_theta*(pi/180)
        /2);
    noise_Delta_q0_raw = cos(noise_Delta_theta*(pi/180)/2);
    norm_noise_Delta_q_raw = norm([noise_Delta_q_raw' noise_Delta_q0_raw
        ]);
    noise_Delta_q(:,j) = noise_Delta_q_raw/norm_noise_Delta_q_raw;
    noise_Delta_q0(j) = noise_Delta_q0_raw/norm_noise_Delta_q_raw;

    % Obtain noise-corrupted measurement quaternions
    [Delta_quat_m_raw, Delta_quat0_m_raw] = qprod(noise_Delta_q(:,j),
        noise_Delta_q0(j), Delta_quat(:,j), Delta_quat0(j));
    norm_Delta_quat_m_raw = norm([Delta_quat_m_raw' Delta_quat0_m_raw]);
    Delta_quat_m(:,j) = Delta_quat_m_raw/norm_Delta_quat_m_raw;
    Delta_quat0_m(j) = Delta_quat0_m_raw/norm_Delta_quat_m_raw;
%     Delta_quat_m(:,j) = Delta_m/mag_Delta_quat_m;
%     Delta_quat0_m(j) = Delta_quat0(j)/mag_Delta_quat_m;
    R_Delta_m_raw(:,:,j) = q2rotmat(Delta_quat_m(:,j),Delta_quat0_m(j));

    % Check Star Tracker noise angle
    [check_noise_Delta_q_raw, check_noise_Delta_q0_raw] = qprod(
       Delta_quat_m(:,j), Delta_quat0_m(:,j), -Delta_quat(:,j),
       Delta_quat0(j));
    norm_check_n_Delta_q = norm([check_noise_Delta_q_raw'
       check_noise_Delta_q0_raw]);
    check_n_Delta_q = check_noise_Delta_q_raw/norm_check_n_Delta_q;
    check_n_Delta_q0 = check_noise_Delta_q0_raw/norm_check_n_Delta_q;
    check_n_Delta_theta_s(j) = 2*asin(norm(check_n_Delta_q))*180/pi;
    check_n_Delta_theta_c(j) = 2*acos(check_n_Delta_q0)*180/pi;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %   Define Star Tracker Alignment Errors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    for k=1:n_lambda,
       lambda = (k-1)*30;
       for l=1:n_phi,
          for m=1:n_epsilon

             epsilon = m*50;

             %%%%%%%%%%%%%%%%%%%%%%
             %   Test parameters
             %%%%%%%%%%%%%%%%%%%%%%

             %phi = (l-1)*0.125 - 89.5;
             phi = (l-1)*20 - 89.5;
             if phi > 89.5,
                phi = 89.5;
             end;
             if phi < -89.5,
                phi = -89.5;
             end;

             cl = cos(lambda*pi/180);
             sl = sin(lambda*pi/180);
             cp = cos(phi*pi/180);
             sp = sin(phi*pi/180);
             ce = cos(epsilon*pi/180);
             se = sin(epsilon*pi/180);

             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
             %   Calculation Setup - assume no errors in Gamma, Delta
             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

             % Calculate M__Up_SC
             M__Up_SD(1,1) = -ce*sl - se*sp*cl;
             M__Up_SD(1,2) = ce*cl - se*sp*sl;
             M__Up_SD(1,3) = se*cp;
             M__Up_SD(2,1) = se*sl - ce*sp*cl;
             M__Up_SD(2,2) = -se*cl - ce*sp*sl;
             M__Up_SD(2,3) = ce*cp;
             M__Up_SD(3,1) = cp*cl;
             M__Up_SD(3,2) = cp*sl;
             M__Up_SD(3,3) = sp;

             %            Phi(1,1) = -sl;
             %            Phi(1,2) = cl;
             %            Phi(1,3) = 0;
             %            Phi(2,1) = - sp*cl;
             %            Phi(2,2) = - sp*sl;
             %            Phi(2,3) = cp;
             %            Phi(3,1) = cp*cl;
             %            Phi(3,2) = cp*sl;
             %            Phi(3,3) = sp;


             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
             %   Calculate quaternion Gamma
             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

             R_Delta_tot(:,:,j) = R_Delta__Body_sST*R_Delta_m_raw(:,:,j
                );
             %R_Gamma_raw = R_Gamma__Body_sAcc'*R_Delta_tot(:,:,j)*
                R_Omega__SD_I'*Phi';
             R_Gamma_raw = R_Gamma__Body_sAcc'*R_Delta_tot(:,:,j)*
                R_Omega__SD_I'*M__Up_SD';
             Gamma_vec_raw = R_Gamma_raw*Acc_down; % Acc measurement
                vector
             Up_vec = -Gamma_vec_raw;
             Gamma_uvec_raw = Gamma_vec_raw/norm(Gamma_vec_raw);
             Up_uvec_raw = -Gamma_uvec_raw;

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

             % Check reality of data angle of Up_vec with respect to
                star
             % tracker data
             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


             Beta_m_raw = acos(Up_uvec_raw'*Body_z)*180/pi; % "sight"
                cone (deg)
             if abs(Beta_m_raw) > Beta    % discard "unseeable" data
                continue;
                %               else % keep reasonable data (counter
                    d)
                %            elseif phi > -80
```

```
    %               continue;
else % keep reasonable data (counter d)
    Beta_m(d) = Beta_m_raw;
    R_Delta(:,:,d) = R_Delta_tot(:,:,j);
    R_Gamma(:,:,d) = R_Gamma_raw;
    %               Gamma_uvec(:,d) = Gamma_uvec_raw;
    Up_uvec(:,d) = Up_uvec_raw;
    lambda_true(d) = lambda;
    phi_true(d) = phi;
    epsilon_true(d) = epsilon;
    R_Delta_m(:,:,d) = R_Delta_m_raw(:,:,j);
    Gamma_uvec(:,d) = Gamma_uvec_raw;
    noise_Delta_quat(:,d) = noise_Delta_q(:,j);
    noise_Delta_quat0(d) = noise_Delta_q0(j);
    noise_Delta_angle(d) = 2*asin(norm(noise_Delta_quat(:,
       d)))*180/pi;

    %  Determine minimum and maximum lambda and phi
    if lambda > max_lambda
       max_lambda = lambda;
    end;
    if lambda < min_lambda
       min_lambda = lambda;
    end;
    if phi > max_phi
       max_phi = phi;
    end;
    if phi < min_phi
       min_phi = phi;
    end;

    Moon_x_m(d) = R_moon*cos(lambda*pi/180)*cos(phi*pi
       /180);
    Moon_y_m(d) = R_moon*sin(lambda*pi/180)*cos(phi*pi
       /180);
    Moon_z_m(d) = R_moon*sin(phi*pi/180);


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

    % Define Accelerometer alignment and measurement
       errors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    %               noise_Gamma_vec = mag_noise_Gamma*(
       randn(3,1) - 0.5); % evenly
    noise_Gamma_vec = mag_noise_Gamma*(randn(3,1) - 0.5);
       % evenly
    %               noise_Gamma_quat = mag_noise_Gamma*(2*
       randn(3,1) - 1); % normaly
```

```
                    Gamma_vec_m(:,d) = Gamma_vec_raw + noise_Gamma_vec;
                    Gamma_uvec_m(:,d) = Gamma_vec_m(:,d)/norm(Gamma_vec_m
                        (:,d));
                    check_n_Gamma_theta(d) = acos(Gamma_uvec(:,d)'*
                        Gamma_uvec_m(:,d))*180/pi;

                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % Define alignment and measurement errors
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %
                    %                R_Delta_m = R_Delta;
                    %                Gamma_uvec_m = Gamma_uvec;

                end;

                d=d+1;

            end;
        end;

    end;

    count(:,j) = [d-old_d];
%
%     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     % Define Star Tracker alignment and measurement errors
%     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%     Delta_quat(:,j)
%     mag_noise_Delta = 1e-6;
%     noise_Delta_quat = mag_noise_Delta*(2*rand(3,1) - 1);
%     Delta_m = Delta_quat(:,j) + noise_Delta_quat;
%     mag_Delta_quat_m = norm([Delta_m' Delta_quat0(j)]);
%     Delta_quat_m(:,j) = Delta_m/mag_Delta_quat_m;
%     Delta_quat0_m(j) = Delta_quat0(j)/mag_Delta_quat_m;
%     R_Delta_m(:,:,j) = q2rotmat(Delta_quat_m(:,j),Delta_quat0_m(j));

%     R_Delta_m = R_Delta;
%     Gamma_uvec_m = Gamma_uvec;

end;

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % Define alignment and measurement errors
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% R_Delta_m = R_Delta;
% Gamma_uvec_m = Gamma_uvec;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Extract lambda, phi, eta - main part of program
```

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Loop to check all good data (counter=i)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:d-1,

    % Given: quaternion Delta, unit vector Gamma
    % R_Delta_m = R_Delta;

%     Acc_meas = Gamma_uvec_m(:,i);
    Acc_meas = Gamma_uvec_m(:,i);
    Gamma_theta = acos((-Acc_meas)'*Body_z);    % radians
    Gamma_cross = cross((-Acc_meas),Body_z);
    Gamma_eta = Gamma_cross/norm(Gamma_cross);
    Gamma_quat = Gamma_eta*sin(Gamma_theta/2);
    Gamma_quat0 = cos(Gamma_theta/2);
    R_Gamma_m(:,:,i) = q2rotmat(Gamma_quat,Gamma_quat0);

    %  Calculate quat[inv(Gamma)*Delta]
    R_invGamma_Delta(:,:,i) = R_Gamma_m(:,:,i)'*R_Delta_m(:,:,i);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %  Extraction of lambda, phi, epsilon from measurements
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    lambda_mR_raw = atan(R_invGamma_Delta(3,2,i)/R_invGamma_Delta(3,1,i)
       )*180/pi;
    phi_mR(i) = asin(R_invGamma_Delta(3,3,i))*180/pi;
    test_ep(i) = epsilon_true(i);
    epsilon_mR_raw(i) = atan(R_invGamma_Delta(1,3,i)/R_invGamma_Delta
       (2,3,i))*180/pi;

    %            % Check of quadrant of lambda
    %
    %            cl = cos(lambda_mR_raw*pi/180);
    %            sl = sin(lambda_mR_raw*pi/180);
    %            cp = cos(phi_mR(j,i)*pi/180);
    %            sp = sin(phi_mR(j,i)*pi/180);
    %            ce = cos(epsilon_mR(j,i)*pi/180);
    %            se = sin(epsilon_mR(j,i)*pi/180);

    % Check and correct for errors in lambda, 90 <= lambda <= 270
    if R_invGamma_Delta(3,1,i) < 0
       lambda_mR_check = lambda_mR_raw + 180;
    else
       lambda_mR_check = lambda_mR_raw;
    end;

    if R_invGamma_Delta(2,3) < 0
```

```
        epsilon_mR_check = epsilon_mR_raw(i) + 180;
    else
        epsilon_mR_check = epsilon_mR_raw(i);
    end;

    % Account for full rotation in lambda, i.e. for lambda >= 360
    if lambda_mR_check < 0
        lambda_mR(i) = lambda_mR_check + 360;
    else
        lambda_mR(i) = lambda_mR_check;
    end;

    if epsilon_mR_check < 0
        epsilon_mR(i) = epsilon_mR_check + 360;
        epsilon_mR_check;
    else
        epsilon_mR(i) = epsilon_mR_check;
    end;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %   Calculate errors in lambda and phi
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    error_lambda(i) = lambda_true(i) - lambda_mR(i);
    error_phi(i) = phi_true(i) - phi_mR(i);
    error_epsilon(i) = epsilon_true(i) - epsilon_mR(i);

%     % Account for full rotation in error_lambda >= 360
%     if error_lambda(i) < -350
%         error_lambda(i) = -(error_lambda(i) + 360);
%     elseif error_lambda(i) > 360
%         error_lambda(i) = error_lambda(i) - 360;
%     end;

    % Check for errors in lambda and phi
    %               if abs(error_lambda(i)) >4e-4 | abs(error_phi(i)) >5
        e-3


    if abs(error_lambda(i)) > 270
        error_lambda(i) = asin(sin(error_lambda(i)/180*pi))*180/pi;
    end;

    if abs(error_epsilon(i)) > 270
        error_epsilon(i) = asin(sin(error_epsilon(i)/180*pi))*180/pi;
    end;


    if error_epsilon(i) < -180
        error_epsilon(i) = error_epsilon(i) + 360;
    end;
```

```
error_horiz(i) = (error_lambda(i)/360)*(R_moon*cos(phi_true(i)*pi
   /180))*(2*pi); % meters
error_vert(i) = error_phi(i)/180*R_moon*pi; % meters
error_radius(i) = sqrt(error_horiz(i)^2 + error_vert(i)^2); % meters
error_rad_angle(i) = atan(error_horiz(i)/error_vert(i))*180/pi;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% More error data for plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if error_horiz(i) < 0
   error_rad_angle(i) = error_rad_angle(i) + 180;
   error_radius_long(i) = -error_radius(i);
else
   error_radius_long(i) = error_radius(i);
end;

if error_vert(i) < 0
   error_radius_lat(i) = -error_radius(i);
else
   error_radius_lat(i) = error_radius(i);
end;


if error_radius(i) > 50 % maximum error radius = 50m
   l_mR(e) = lambda_mR(i);
   p_mR(e) = phi_mR(i);
   err_l(e) = error_lambda(i);
   err_p(e) = error_phi(i);
   l_true(e) = lambda_true(i);
   p_true(e) = phi_true(i);
   e_mR(e) = epsilon_mR(i);
   Del_q_m(:,e) = Delta_quat_m(:,j);
   Del_q0_m(:,e) = Delta_quat0_m(:,j);
   R_Del_m(:,:,e) = R_Delta_m(:,j);
   e_rad(e) = error_radius(i);
   e_hor(e) = error_horiz(i);
   e_ver(e) = error_vert(i);

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %  Plot Navigation key errors on lunar surface plot
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   ENav_x_m(1,e) = R_moon*cos(lambda_mR(i)*pi/180)*cos(phi_mR(i)*pi
      /180);
   ENav_y_m(1,e) = R_moon*sin(lambda_mR(i)*pi/180)*cos(phi_mR(i)*pi
      /180);
   ENav_z_m(1,e) = R_moon*sin(phi_mR(i)*pi/180);

   ENav_x_m(2,e) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*cos(
      lambda_mR(i)*pi/180)*(1 + error_radius(i)/R_moon*rad_fac)*cos
```

```
            (phi_mR(i)*pi/180);
        ENav_y_m(2,e) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(
            lambda_mR(i)*pi/180)*(1 + error_radius(i)/R_moon*rad_fac)*cos
            (phi_mR(i)*pi/180);
        ENav_z_m(2,e) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(
            phi_mR(i)*pi/180);

        e = e + 1;

    end;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %   Plot Navigation errors on lunar surface plot
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    eNav_x_m(1,i) = R_moon*cos(lambda_mR(i)*pi/180)*cos(phi_mR(i)*pi
        /180);
    eNav_y_m(1,i) = R_moon*sin(lambda_mR(i)*pi/180)*cos(phi_mR(i)*pi
        /180);
    eNav_z_m(1,i) = R_moon*sin(phi_mR(i)*pi/180);

    eNav_x_m(2,i) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*cos(
        lambda_mR(i)*pi/180)*(1 + error_radius(i)/R_moon*rad_fac)*cos(
        phi_mR(i)*pi/180);
    eNav_y_m(2,i) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(
        lambda_mR(i)*pi/180)*(1 + error_radius(i)/R_moon*rad_fac)*cos(
        phi_mR(i)*pi/180);
    eNav_z_m(2,i) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(
        phi_mR(i)*pi/180);

    i = i + 1;
end;

tot_poss_iter = n*n_lambda*n_phi*n_epsilon;
tot_iter = d - 1;

star_tracker_n = n
frac_iter = tot_iter/tot_poss_iter

mu = mean(error_radius)
sig = std(error_radius)
% mag_noise_Gamma
% mag_noise_Delta_theta

% max_lambda
% min_lambda
% max_phi
% min_phi

% Up_angle = acos(Up_uvec'*[0 0 1]')*180/pi;
% mu_Up_angle = mean(acos(Up_uvec'*[0 0 1]')*180/pi)
% [noise_Delta_angle(d-1) mean(check_n_Gamma_theta) std(
    check_n_Gamma_theta)]

ci_limit = mu + 3*sig;

ci_limit_stat = [mu sig ci_limit];
```

A.2 Current CelNav Algorithm

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

%                   confirms calculations of lambda, phi
%
%          - calculates Gamma ("truth" data) given Delta and SC
%          - yields: unit vector Gamma
%                    unit quaternion Delta
%          - confirms normalizations of quaternions
%          - recalculates lambda, phi, and epsilon given Gamma and Delta
%          - calls rand_q.m, xprod_mat.m, qprod.m, q2rotmat.m
%          - generates vectors of data
%          - test for lander "sight" viewing angle (Beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Variable Definitions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% R_x   = rotation matrix for x
% x_vec = vector for x
%
% eta   = unit vector (defined for quaternion)
% thetha = rotation angle (defined for quaternion)
%
% Gamma --> Accelerometer
% Delta --> Star Tracker
% Phi   --> M__ENU_SD (function of lambda and phi)
% Psi   --> Accelerometer Output
% Omega --> Moon
%
% n       = number of star tracker measurements
% n_lambda = iterations of lambda (full 0 to +360 degree coverage)
% n_phi   = iterations of phi (full -90 to +90 degree coverage)
% d - 1   = number of acceptable data points
%
% Beta = lander "sight" angle (not boresight angle) according to 30 deg
%          crater slope
%
% "lambda" = longitude
% "phi"    = lattitude
% "Heading" = heading
% "alpha"  = tilt
% "beta"   = slope
%
% lambda_true = lambda truth data
% phi_true   = phi truth data
% lambda_mR  = extracted lambda
% phi_mR     = extracted phi
%
```

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

%                          Monte Carlo Simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Clears Data after every run
   clc
   close all
   clear all

for phi=0;
% Generates Folders to be Used in Data Sorting
   folders(phi)

%Beginning of Monte Carlo Loop

for lambda =0;

   alpha = 30;                 % Tilt
   beta  = 2;                  % Slope
   alignment_error_star= 60;   % Arcsec (Error Star)
   alignment_error_Acc = 60;   % Arcsec (Error Accelerometer)
   mag_noise_Gamma   = 1e-6;  % Magnitude

% Body Labels
   Body_z = [0 0 1]';

% Counters for Number of runs to be performed during each iteration
   d = 1;
   n = 1;
   p = 1;
   n_lambda = 13;
   n_phi    = 12;
   n_epsilon = 13;

% Define Accelerometer alignment
   Acc1_vec_r = [1 0 0]';% Acc x-vector
   Acc2_vec_r = [0 1 0]';% Acc y-vector
   Acc3_vec_r = [0 0 1]';% Acc z-vector
   Acc1_vec  = Acc1_vec_r/norm(Acc1_vec_r);% Acc x-vector normalized
   Acc2_vec  = Acc2_vec_r/norm(Acc2_vec_r);% Acc y-vector normalized
   Acc3_vec  = Acc3_vec_r/norm(Acc3_vec_r);% Acc z-vector normalized

% Define M__A_Body with respect to Acc1

% Creates Scalar Portion of Quaternion Vector
   M__A1_Body_eta = cross(Body_z,Acc1_vec);
% Degrees
   M__A1_Body_theta = acos(Body_z'*(Acc1_vec))*180/pi;
% Quaternion Vector Body to Acc z-vector
   M__A1_Body_quat = M__A1_Body_eta*sin(M__A1_Body_theta/2*pi/180);
```

```
% Quaternion Scalar Body to Acc z-vector
   M__A1_Body_quat0 = cos(M__A1_Body_theta/2*pi/180);
% Total Quaternion
   Q_Acc_1        = [M__A1_Body_quat(1),M__A1_Body_quat(2),
      M__A1_Body_quat(3),M__A1_Body_quat0]';
% Normalizes Acc1 Quaternion
   Q_Acc_1_n      = Q_Acc_1/norm(Q_Acc_1);
%Rotation Matrix
   R_M__A1_Body   = Q2A(Q_Acc_1_n);

% Define M__A_Body with respect to Acc2
% Creates Scalar Portion of Quaternion Vecter
   M__A2_Body_eta = cross(Body_z,Acc2_vec);
% Degrees
   M__A2_Body_theta = acos(Body_z'*(Acc2_vec))*180/pi;
% Quaternion Vector Body to Acc z-vector
   M__A2_Body_quat = M__A2_Body_eta*sin(M__A2_Body_theta/2*pi/180);
   M__A2_Body_quat0 = cos(M__A2_Body_theta/2*pi/180);
% Total Quaternion Matrix
   Q_Acc_2        = [M__A2_Body_quat(1),M__A2_Body_quat(2),
      M__A2_Body_quat(3),M__A2_Body_quat0]';
% Normalizes Acc2 Quaternion
   Q_Acc_2_n      = Q_Acc_2/norm(Q_Acc_2);
% Rotation Matrix
   R_M__A2_Body   = Q2A(Q_Acc_2);

% Define M__A_Body with respect to Acc3
% Creates Scalar Portion of Quaternion Vecter
   M__A3_Body_eta = cross(Body_z,Acc3_vec);
% Degrees
   M__A3_Body_theta = acos(Body_z'*(Acc3_vec))*180/pi;
% Quaternion Vector Body to Acc z-vector
   M__A3_Body_quat = M__A3_Body_eta*sin(M__A3_Body_theta/2*pi/180);
   M__A3_Body_quat0 = cos(M__A3_Body_theta/2*pi/180);
% Total Quaternion
   Q_Acc_3        = [M__A3_Body_quat(1),M__A3_Body_quat(2),
      M__A3_Body_quat(3),M__A3_Body_quat0]';
% Normalizes Acc3 Quaternion
   Q_Acc_3_n      = Q_Acc_3/norm(Q_Acc_3);
% Rotation Matrix
   R_M__A3_Body   = Q2A(Q_Acc_3);

% Define Star tracker alignment
% Star tracker z-vector
   neg_Star_tracker_vec    = [0 0 1]';
% Star tracker z-(unit) vector
   neg_Star_tracker_vec_unit = neg_Star_tracker_vec/norm(
      neg_Star_tracker_vec);
% Creates Scalar Portion of Quaternion Vecter
   M__C_Body_eta           = cross((Body_z),neg_Star_tracker_vec_unit);
% Degrees
```

```
   M__C_Body_theta        = acos(neg_Star_tracker_vec_unit'*Body_z);
% Quaternion Vector Body to Acc z-vector
   M__C_Body_quat         = M__C_Body_eta*sin(M__C_Body_theta/2*pi/180)
       ;
   M__C_Body_quat0        = cos(M__C_Body_theta/2*pi/180);
% Total Quaternion
   Q_star                 = [M__C_Body_quat(1),M__C_Body_quat(2),
       M__C_Body_quat(3),M__C_Body_quat0]';
% Normalize Star Tracker Alignment Quaternion
   Q_star_n               = Q_star/norm(Q_star);
% Rotation Matrix
   R_M__C_Body            = Q2A(Q_star_n);
% Alignment Matrix
   R_M__C_Body_ae         = RR_b(alignment_error_star);

% Define lunar coordinate transformation
% Moon Radius(Meters)
   R_moon                 = 1738.2*1e3;
% NASA Lunar Models
   Omega__SC_I            = eye(3);
   Omega__SD_SC           = eye(3);
% Total Model
   Omega__SD_I            = Omega__SD_SC*Omega__SC_I;
% Alignment Model
   R_Omega__SD_I_E        = eye(3);
% Needs to be updated with NASA lunar model and need earth geoide model
   for testing
   R_ Omega__SD_I          = Omega__SD_I;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate random star tracker quaternion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Body reference frame
   body = [1 0 0;0 1 0;0 0 1];

% Quaternion Generator "True" Star Tracker Quaternion
   [Delta_quat,Delta_quat0] = rand_q(n);

% Generates Rotation Matrix(Extracts, Normalize, Create Rotation)
   Q_Delta     = [Delta_quat(1),Delta_quat(2),Delta_quat(3),Delta_quat0
       ]';
   Q_Delta_n   = Q_Delta/norm(Q_Delta);
   R_Delta__I_C = Q2A(Q_Delta_n);

for j=1:n

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Define Star Tracker measurement errors
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Noise in Arc seconds (X,Y,Z Axes)
    noise_x = 10;
    noise_y = 10;
    noise_z = 40;

% Raw Quaternion for NASA Star Tracker Model
    Delta_raw = [1,Delta_quat0(j),Delta_quat(1,j,1),Delta_quat(2,j,1)
        ,Delta_quat(3,j,1)];

% NASA Star Tracker Noise Model
    [Delta_quat_n,flag,frame] = standard_ast_model(Delta_raw,noise_x,
        noise_y,noise_z,body);

% Quaternion Extraction and Normalization
    Q_Startracker  = [Delta_quat_n(3),Delta_quat_n(4),Delta_quat_n(5)
        ,Delta_quat_n(2)]';
    Q_Startracker_n = Q_Startracker/norm(Q_Startracker);

 % Generates a Rotational Matrix from a quaternion
    R_Delta__I_C_m = Q2A(Q_Startracker_n);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Generate test data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
old_d = d;
old_p = p;

% Longitued -180->180
for k=1:n_lambda,

    for l=1:n_phi,

        for m=1:n_epsilon
            % Heading
                epsilon = (m-1)*15;

            %Generate ST(alpha, beta)
                cb = cos(beta*pi/180);
                sb = sin(beta*pi/180);
                ca = cos(alpha*pi/180);
                sa = sin(alpha*pi/180);

            %Actual ST Matrix
                ST = [ cb sb*sa -sb*ca;
                       0    ca    sa;
                      sb -cb*sa cb*ca];

            %Gamma Rotation Matrices
            % Note in actual practice all three 3D accelerometers
                will be needed
                R_Gamma1 = ST'*R_M__A1_Body';
```

```
     R_Gamma2 = ST'*R_M__A2_Body';          % Not Used
     R_Gamma3 = ST'*R_M__A3_Body';          % Not Used

%Sets up sin and cos for 'true' rotation matrix
    cl = cos(lambda*pi/180);                % Cosine
       Lambda
    sl = sin(lambda*pi/180);                % Sine Lambda
    cp = cos(phi*pi/180);                   % Cosine Phi
    sp = sin(phi*pi/180);                   % Sine Phi
    ce = cos(epsilon*pi/180);               % Cosine
       Epsilon
    se = sin(epsilon*pi/180);               % Sine Epsilon

% Heading Matrix
    He__Down_NED = [ce se 0; -se ce 0; 0 0 1];

% Latitude and Longitude
    LL__ENU_SD  = [-sl cl 0; -cl*sp -sl*sp cp; cl*cp sl*cp
        sp];

% Conversion Matrix from ENU to SD to NED to ENU
    U__NED_ENU  = [0 1 0; 1 0 0; 0 0 -1];

% Generates 'truth' model
    Phi__Down_SD = He__Down_NED*U__NED_ENU*LL__ENU_SD;

% Generates Delta 'truth' model
    Delta = Omega__SD_I'*Phi__Down_SD'*R_Gamma3*
       R_M__A3_Body*R_M__C_Body;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Calculate quaternion Gamma
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generates Upsilon model
    Upsilon   = (Omega__SD_I*Delta*R_M__C_Body)'*(
       U__NED_ENU*LL__ENU_SD)';

% Extracts alpha, beta, epsilon from upsilon matrix
% Extracts Tilt
    alpha_U   = asind(Upsilon(2,3));
    alpha_U_1 = atan2(Upsilon(2,3),...
                (sqrt(Upsilon(2,1)^2+Upsilon(2,2)^2)))
                   *180/pi;
% Extracts Slope
    beta_U    = atan2(-Upsilon(1,3),Upsilon(3,3))*180/pi;

% Generates 'truth' gamma model
    R_Gamma__Down_A3_r = Phi__Down_SD*R_Omega__SD_I*...
                         R_Delta__I_C_m*R_M__C_Body*...
                         R_M__A3_Body';
```

```
% Generates Gamma3 Vector(1,2,3)
    Gamma_vec1_raw = R_Gamma__Down_A3_r*Acc1_vec;
    Gamma_vec2_raw = R_Gamma__Down_A3_r*Acc2_vec;
    Gamma_vec3_raw = R_Gamma__Down_A3_r*Acc3_vec;

% Generates Phi Test Matrix
    Phi__Down_SD_test(:,:,d) = Phi__Down_SD;

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

%  Check reality of data angle of Up_vec with respect to
    star
%  tracker data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Setting True Latitude(Lambda), Longitude(Phi) and
        Heading(Epsilon) Data
        lambda_true(p) = lambda;
        phi_true(p)    = phi;
        epsilon_true(p) = epsilon;

    % Sets up True Gamma Vector and Normalizes Each Vector
        R_Gamma__Down_A3r(:,:,p) = R_Gamma__Down_A3_r;
        Gamma_vec1(:,d) = Gamma_vec1_raw/norm(
            Gamma_vec1_raw);
        Gamma_vec2(:,d) = Gamma_vec2_raw/norm(
            Gamma_vec2_raw);
        Gamma_vec3(:,d) = Gamma_vec3_raw/norm(
            Gamma_vec3_raw);

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

    % Define Accelerometer alignment and measurement
        errors
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        % Generates Noisy Gamma Vectors (1,2,3) Evenly
            Distributed
            noise_Gamma_vec1 = mag_noise_Gamma*(randn(3,1)
                - 0.5);
            noise_Gamma_vec2 = mag_noise_Gamma*(randn(3,1)
                - 0.5);
            noise_Gamma_vec3 = mag_noise_Gamma*(randn(3,1)
                - 0.5);

        % Generates Measured Gamma Vectors (1,2,3)(True +
            Noise)
```

```
                      Gamma_vec1_m = Gamma_vec1(:,d) +
                         noise_Gamma_vec1;
                      Gamma_vec2_m = Gamma_vec2(:,d) +
                         noise_Gamma_vec2;
                      Gamma_vec3_m = Gamma_vec3(:,d) +
                         noise_Gamma_vec3;

                   % Generates Normalized Gamma Vectors (1,2,3)
                      Gamma_uvec1_m(:,d) = Gamma_vec1_m/norm(
                         Gamma_vec1_m);
                      Gamma_uvec2_m(:,d) = Gamma_vec2_m/norm(
                         Gamma_vec2_m);
                      Gamma_uvec3_m(:,d) = Gamma_vec3_m/norm(
                         Gamma_vec3_m);

                   % Counters
                      p = p+1;
                      d = d+1;
     end;
  end;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Extract lambda, phi, epsilon - main part of program
%            Solver Part of CelNAV Algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=old_p:p-1,

  % Assigns Accelerometer Vector (1,2,3) from Gamma Vector 1,2,3
     Acc1_m(:,i) = Gamma_uvec1_m(:,i);
     Acc2_m(:,i) = Gamma_uvec2_m(:,i);
     Acc3_m(:,i) = Gamma_uvec3_m(:,i);

  % Assembles Accelerometer Rotation Matrix from Accelerometer
     Vectors
     R_Gamma__Down_A3_test(:,:,i) = [Acc1_m(:,i) Acc2_m(:,i)
        Acc3_m(:,i)];

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

   %                  Calculate Phi__Down_SD_e
   %         (extracted Phi from measurements Delta and Gamma)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % Creates Accelerometer Alignment Error Matrix
     R_Error_Body = RR_b(alignment_error_Acc);

   % Gamma Rotation Matrix with Noise Addition
     Phi__Down_SD_e(:,:,i) = R_Gamma__Down_A3_test(:,:,i)*...
                          R_Error_Body*R_M__A3_Body*...
                         R_M__C_Body'*R_M__C_Body_ae'*...
                         R_Delta__I_C_m'*...
                         R_Omega__SD_I'*R_Omega__SD_I_E';

% Generates Latitude
    lambda_mR(i) = atan2(-Phi__Down_SD_e(3,2,i),-Phi__Down_SD_e
        (3,1,i))*180/pi;

% Generates Longitude
    if -Phi__Down_SD_e(3,3,i)>1
        phi_mR(i) = 90;
    elseif -Phi__Down_SD_e(3,3,i)<-1
        phi_mR(i)= -90;
    else
        phi_mR(i) = atan2(-Phi__Down_SD_e(3,3,i),(sqrt(
            Phi__Down_SD_e(2,3,i)^2+Phi__Down_SD_e(1,3,i)^2)))
            *180/pi;
    end;

% Generates Heading
    epsilon_mR(i) = atan2(-Phi__Down_SD_e(2,3,i),Phi__Down_SD_e
        (1,3,i))*180/pi;

% Generates Error Maticies for Lambda, phi and epsilon
    error_lambda(i) = lambda_true(i) - lambda_mR(i);
    error_phi(i)    = phi_true(i)   - phi_mR(i);
    error_epsilon(i) = epsilon_true(i)- epsilon_mR(i);

% Generated Errors in degrees
    error_alpha_Upsilon = alpha-alpha_U;
    error_alpha_Upsilon1 = alpha-alpha_U_1;
    error_beta_Upsilon = beta-beta_U;

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

% Account for greater than 360 lambda and epsilon error
%                   (If available)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Error Lambda
    if error_lambda(i) >= 350
        error_lambda(i) = error_lambda(i) - 360;
    elseif error_lambda(i) <= -350
        error_lambda(i) = error_lambda(i) + 360;
    end;

% Error Epsilon
    if abs(error_epsilon(i)) >= 350
        error_epsilon(i) = error_epsilon(i) - 360;
    elseif error_epsilon(i) <= -350
```

```
                error_epsilon(i) = error_epsilon(i) + 360;
            end;

        % Note: error_radius will only work if errors are small.
        % Gernerates the error result in Latitude, Longitute and heading
            error_horiz(i)    = (error_lambda(i)/360)*(R_moon*cos(phi_true
                (i)*pi/180))*(2*pi); % meters
            error_vert(i)     = error_phi(i)/180*R_moon*pi;
                                                % meters
            error_radius(i)   = sqrt(error_horiz(i)^2 + error_vert(i)^2);
                                 % meters
            error_rad_angle(i) = atan2(error_horiz(i),error_vert(i))*180/
                pi;             % degrees

        end;
    end;

% End of Monte Carlo for Loop
% Beginning of Data Recording
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%         Creates Files for Data Recording and Changes Directory    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Creating Dynamic File Names for CelNav Data
        s = sprintf('24-June-09 Stocastic Mean Phi = %d Lambda = %d',phi,
            lambda);

    % Creating Dynamic File Names for Stocastic Data
        s1 = sprintf('24-June-09 Stocatsic Phi=%d Lambda=%d Results',phi,
            lambda);

    % Changes Working Directory
        %Folder3 = sprintf('C:\\Documents and Settings\\controls\\Desktop
            \\CelNav\\CelNav Versions\\NASA Rework with Tilt Slope -
            Current\\Run Results\\Monte Carlo Results\\Phi = %d\\Data',
            phi);
        %cd(Folder3);

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

%                   Generates Data For Sigma Circles                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


   % Returns to Actual Working Directory
        cd('C:\Documents and Settings\controls\Desktop\CelNav\CelNav
            Versions\NASA Rework with Tilt Slope - Current');

   % Generates Sigma Circle Data
```

```
        [X,Y,X1,Y1,X2,Y2,A,B,r1,r2,r3] = circle(error_horiz,error_vert);

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

%                        Records Stocastic Data                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Changes Working Directory
        Stoc = sprintf('C:\\Documents and Settings\\controls\\Desktop\\
            CelNav\\CelNav Versions\\NASA Rework with Tilt Slope -
            Current\\Run Results\\Monte Carlo Results\\Phi = %d\\
            Stocastic',phi);
        cd(Stoc);

    % Format Variables
        X=X';
        Y=Y';
        X1=X1';
        Y1=Y1';
        X2=X2';
        Y2=Y2';
        A=A';
        B=B';

        Stocast_Sigma=[r1 r2 r3];
        %Stocast_Sigma=[X,Y,X1,Y1,X2,Y2];
        mean=[A,B];

    % Standard Deviation(Sigma-1x)
        save(s, 'mean','-ASCII','-double','-tabs');
        save(s1, 'Stocast_Sigma','-ASCII','-double','-tabs');

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

%                        Generating Error Plot                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                        Plot Latitude and Longitude                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        % Creates Graph of Distance from Actual Location
            h1=figure(1);
            set(h1,'visible','off');
                plot(lambda_mR-lambda,phi_mR-phi,'.');
                grid on
                xlabel('Difference in True to Measured Lambda (Degrees)');
                ylabel('Difference in True to Measured Phi (Degrees)');
                s2 = sprintf('Latitude and Longitude Plot Phi=%d Lambda=%d
                    ',phi,lambda);
                s3 = sprintf('Differnece Phi=%d Acc Align=%d Star Align=%d
                     Acc Noise=%d',phi,alignment_error_Acc,
                    alignment_error_star,mag_noise_Gamma);
                zlabel('Latitude and Longitude Plot');
                title({s2;s3});
                s1 = sprintf('Latitude_and_Longitude_Plot_Phi=%d_Lambda=%d
                    ',phi,lambda);

        % Changes Working Directory
            Folder4 = sprintf('C:\\Documents and Settings\\controls\\
                Desktop\\CelNav\\CelNav Versions\\NASA Rework with Tilt
                Slope - Current\\Run Results\\Monte Carlo Results\\Phi = %
                d\\Error',phi); cd(Folder4);

        % Prints Graph to Current Directory
            print('-f1', '-depsc', s1);

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

%                   Generates 3-Sigma Plots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


        % Creates Graph of Location Error
            h2=figure(2);
            set(h2,'visible','off');
                plot(error_horiz,error_vert,'.r');              % Scatter
                    Plot of Error Data
                hold on;
                idd = plot(A,B,'b');                                %
                    Center
                set ( idd, 'Marker', '*' )
                set ( idd, 'MarkerSize', 20 )
                set ( idd, 'Color', 'Blue' )
                hold on;
                plot(X,Y,'-b');                                 % Sigma-1
                    Circle
                hold on;
                plot(X1,Y1,'-b');                               % Sigma-2
                    Circle
                hold on;
                plot(X2,Y2,'-b');                               % Sigma-3
                    Circle
                hold on;
                id=plot(0,0);
                set ( id, 'Marker', '.' )
                set ( id, 'MarkerSize', 20 )
                set ( id, 'Color', 'Black' )
                grid on;
                hold off;
                axis equal;
```

```
                xlabel('Longitude Navigational Error (meters)');
                ylabel('Latitude Naviagational Error (meters)');
                s6 = sprintf('Corresponds to Phi = %d Lambda = %d',phi,
                   lambda);
                s5 = sprintf('Acc Align = %d Star Align = %d Acc Noise = %
                   d',alignment_error_Acc,alignment_error_star,
                   mag_noise_Gamma);
                ss = sprintf('Mean Location (X=%4.2f,Y=%4.2f)',A,B);
                title({s6;s5;ss});
                s4 = sprintf('Phi=%d_Lambda=%d_Acc_Align=%d_Star_Align=%
                   d_Acc_Noise=1_e_-_6',phi,lambda,alignment_error_Acc,
                   alignment_error_star);

        % Changes Working Directory
            Folder5 = sprintf('C:\\Documents and Settings\\controls\\
               Desktop\\CelNav\\CelNav Versions\\NASA Rework with Tilt
               Slope - Current\\Run Results\\Monte Carlo Results\\Phi = %
               d\\Location',phi);
            cd(Folder5);

        % Prints Graph to Current Directory
            print('-f2', '-depsc', s4);

        % Returns to Actual Working Directory
            cd('C:\Documents and Settings\controls\Desktop\CelNav\CelNav
               Versions\NASA Rework with Tilt Slope - Current');
    end;
end;
```

Appendix B

CelNav Models

B.1 System Model

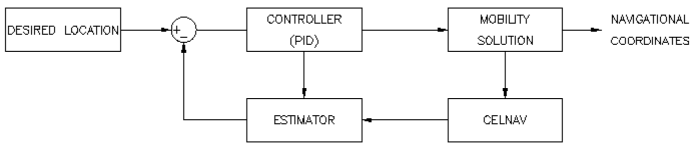

Figure B.1: System Model

B.2 PID

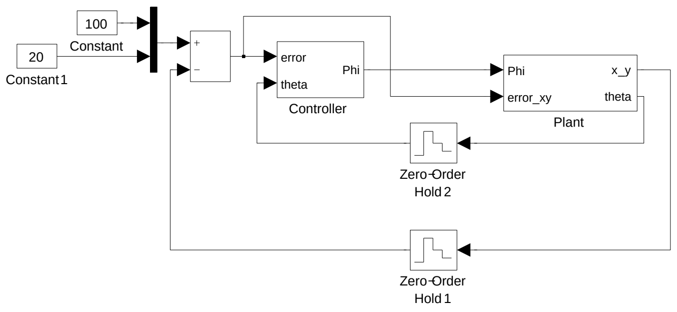

Figure B.2: PID System

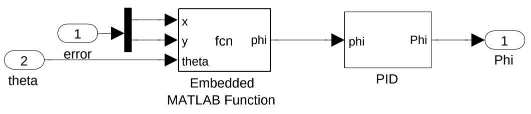

Figure B.3: PID Controller

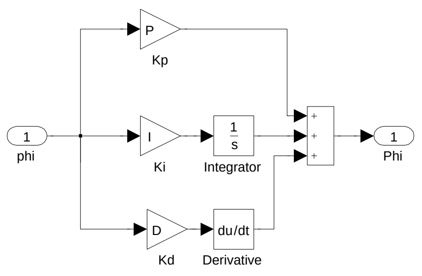

Figure B.4: PID Setup

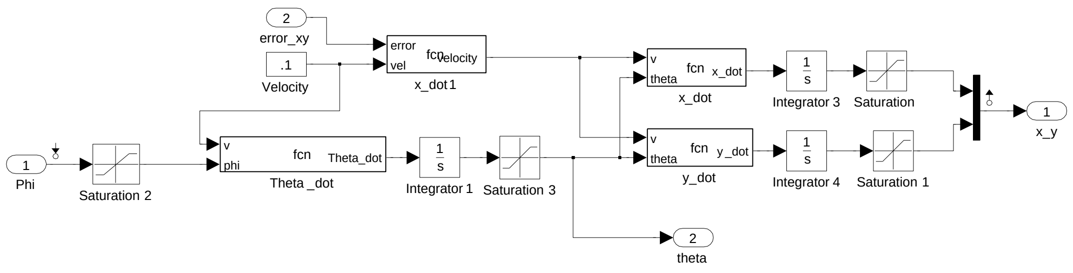

Figure B.5: Plant Setup (Rover Model)

B.2.1 Velocity Determination

```
if((abs(error(1,1)) < .1) && (abs(error(2,1)) < .1))
   velocity = 0;
else
   velocity = vel;
end
```

B.2.2 Theta (Heading)

```
    Theta_dot = v/.5*tan(phi);
```

B.2.3 Velocity x-direction

```
    x_dot = v*cos(theta);
```

B.2.4 Velocity y-direction

```
    y_dot = v*sin(theta);
```

B.2.5 Heading Correction

```
    if(x<=0)
        x=1;
    end

    alpha = atan(y/x);
    phi = alpha-theta;
```

B.3 EKF

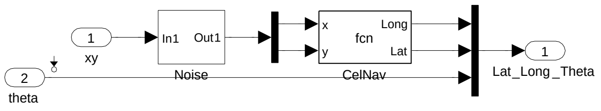

Figure B.6: CelNav Setup

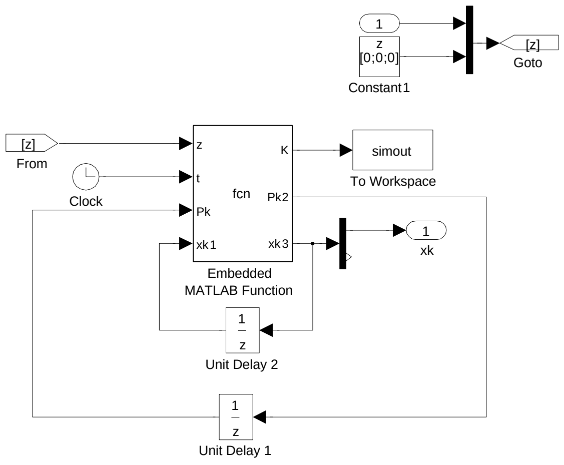

Figure B.7: EKF Setup

```
    if t>.1
        xk = [xk1(1,1) xk1(2,1) xk1(3,1) xk1(4,1) xk1(5,1) xk1(6,1) ];
    else
        xk = [ .1 .1 .1 .1 .1 .1 ];
    end
% Calculate the Jacobians at each time step

F=[     1                     0                      0               0 0
    0;
        0                     1                      0               0 0
            0;
 -h*xk(1,4)*sin(xk(1,3)) h*xk(1,4)*cos(xk(1,3))     1               0 0
     0;
     h*cos(xk(1,3))       h*sin(xk(1,3))       h*tan(xk(1,6))/l    1 0
         0;
        0                     0                      0               h 1
            0;
        0                     0           h*xk(1,4)/l*(1+tan(xk(1,6)^2)) 0
             0 1]';


H =   [1 0 0 0 0 0;
       0 1 0 0 0 0;
       0 0 1 0 0 0;
       0 0 0 0 0 0;
       0 0 0 0 0 0;
       0 0 0 0 0 0];

M=1;

xk2 = [ xk(1,4)*cos(xk(1,3));
       xk(1,4)*sin(xk(1,3));
      xk(1,4)/l*tan(xk(1,6));
           0;
           0;
           0].*h + xk1; % Projected State


    Pk1= F*Pk*F' + L*Q*L'; % Projected Covariance

    K = Pk1*H'/(H*Pk1*H'+M*R*M'); % Kalman gain

    xk3 = xk2+K*(z-H*xk2); %Estimated State

    Pk2 = (eye(6)-K*H)*Pk1*(eye(6)-K*H)'+K*R*K'; %New Covarience Matrix
```

B.4 H-Infinity (H∞)

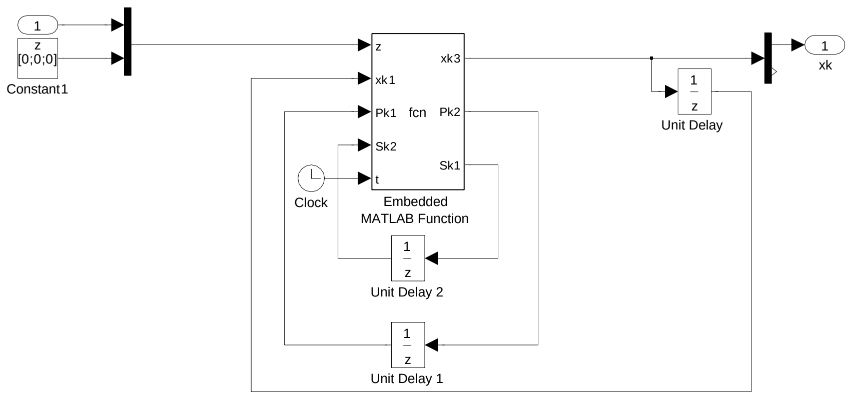

Figure B.8: H-∞ Setup

```
xk = [xk1(1,1) xk1(2,1) xk1(3,1) xk1(4,1) xk1(5,1) xk1(6,1) ];
% Calculate the Jacobians at each time step

A=[     1                     0                      0              0 0
    0;
        0                     1                      0              0 0
           0;
 -h*xk(1,4)*sin(xk(1,3)) h*xk(1,4)*cos(xk(1,3))     1              0 0
     0;
     h*cos(xk(1,3))       h*sin(xk(1,3))       h*tan(xk(1,6))/l    1 0
        0;
        0                     0                      0              h 1
           0;
        0                     0         h*xk(1,4)/l*(1+tan(xk(1,6)^2)) 0
            0 1]';

xkn = [xk1(1,1) xk1(2,1) xk1(3,1) xk1(4,1) xk1(5,1) 0 ]';

% Propagate your EKF equations
Sk =   [10 0 0 0 0 0;
        0 1 0 0 0 0;
        0 0 1 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0;
        0 0 0 0 0 0];

Pk =   [10 0 0 0 0 0;
        0 10 0 0 0 0;
        0 0 10 0 0 0;
        0 0 0 10 0 0;
        0 0 0 0 10 0;
        0 0 0 0 0 10];
```

```
xkx = [0;
       0;
       0;
       0;
       0;
       0];

if t > .1

Sk1 = L'*Sk2*L;

K = Pk1*inv(eye(6) - phi*Sk1*Pk1 + H'*R_inv*H*Pk1)*H'*R_inv; % Kalman
    gain

xk3 = A*xkn+A*K*(z-H*xkn); %Estimated State

Pk2 = A*Pk1/(eye(6)-phi*Sk1*Pk1+H'*R_inv*H*Pk1)*A'+ Q;

else

Sk1 = L'*Sk*L;

K = Pk*inv(eye(6)-phi*Sk1*Pk+H'*R_inv*H*Pk)*H'*R_inv; % Kalman gain

xk3 = A*xkx+A*K*(z-H*xkx); %Estimated State

Pk2 = A*Pk/(eye(6)-phi*Sk1*Pk+H'*R_inv*H*Pk)*A'+Q;

end
```

B.5 Sliding Mode Observer (SMO)

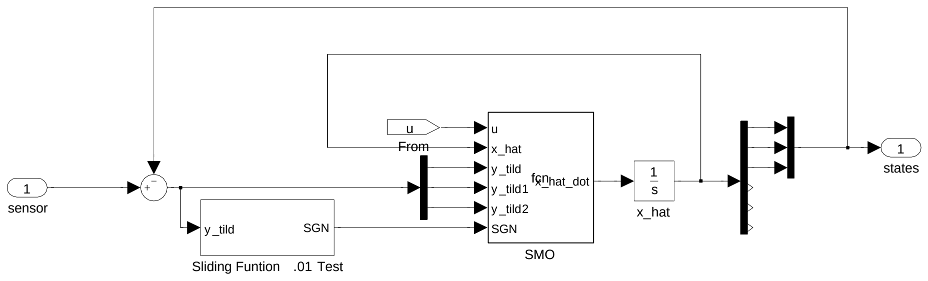

Figure B.9: SMO Setup

```
x=x_hat;
l=5;

x_hat = [x(1,1); x(2,1); x(3,1);0;0;0];

% Calculate the Jacobians at each time step
A_hat = [ x_hat(4,1)*cos(x_hat(3,1));
       x_hat(4,1)*sin(x_hat(3,1));
      x_hat(4,1)/l*tan(x_hat(6,1));
           0;
           0;
           0]; % Projected State

y_tild3 = [y_tild; y_tild1; y_tild2;0;0;0];

%A_hat = A_d*x_hat;

Bu = [0,0;0,0;0,1;0,0;0,0;0,0]*[u(1,1);u(2,1)];

L_ytild = L'*y_tild3;

x_hat_dot = A_hat + Bu + L_ytild + SGN;
```

Appendix C

CelNav Simulation (Experimental)

C.1 Hardware

Multiple different pieces of hardware are used in the construction of a test environment run the simulation using the CelNav algorithm. The main sensors that are used to test the CelNav algorithm are an accelerometer triad, a light sensing array, and a single board computer ro record and distribute all sensor data to a computational mathematics interface. A test enclosure has been constructed for confirmation of the CelNav algorithm. It will be assumed that all hardware will only have orthogonal misalignment errors.

C.1.1 Lunabot

The Lunabot came out of a lunar mining competition that is held by NASA, since 2010. The rover was designed to be light and easily portable. The Lunabot is controlled by remote connection to a laptop. The lunabot is a four (4) wheeled, tank drive rover, this means it can independently control the left or right side wheels. The Lunabot contains an integrated sensor platform, containing the aforementioned sensors, and oriented along the x-y-z body (Lunabot) coordinate frame. Sensor data is fed into a computational mathematics program for analysis; this program will exist on a stand alone computer that will also be attached to the experimental rover. The computer will transmit data to a simulated ground station (laptop). This will allow for monitoring of location as well as the ability for manually control during non-autonomous simulations.

C.1.2 3D - Accelerometer

A 3-D accelerometer is required to accurately determine the Lunabot's current location. Accelerometer triad is a device where there are three separate single axis accelerometers oriented along the X, Y, and Z axes of an integrated sensor. An accelerometer describes body with respect to the selenodetic center of a planetary body, as explained in Chapter I. The accelerometer used here would be of a special type, and would always point down towards the center to the planetoid. The data produced by this accelerometer allows for the extraction of both tilt and slope, which is necessary to fully determine the attitude of the rover or astronaut in relation to the extra-planetary reference frame.

The measurement output from the accelerometer is given as Γ. This output is assumed to be given such that Γ provides the "Down" position with respect to the accelerometer coordinate system, such that:

$$[\ \Gamma\ ]_{Acc}^{Down} \longrightarrow \text{Accelerometer Measurement}$$

C.1.3 Contained Attitude Star Tracking Sensor (CASTS)

The Contained Attitude Star Tracking Sensor (CASTS) is used to calculate a unit quaternion that will indicate the vehicles pointing direction and rotation. CASTS consists of a three camera array positioned on the top of the vehicle and arranged such that each camera has a different pointing direction. The CAST algorithm can be found in Appendix D for further investigation.

The images taken from these cameras are transferred to an algorithm developed by Tyler Wills, a fellow graduate student at the University of New Hampshire. This algorithm transforms known information about the unique images from a database, also developed by Tyler Wills, into an unit quaternion.

CASTS works by matching known color patterns stored in a local database. The patterns consist of red, blue, and green LEDs arranged in patterns of two LEDs per surface. By knowing how far each LED was placed from each other a proper quaternion can be developed. In order to account for inconsistencies in the environment such as blind spots, three stereoscopic cameras, as stated above, will be used. Thus by seeing both sides of the wall or a wall and the ceiling it is possible to get a unique quaternion. The quaternion is defined as followed:

$$q = \begin{bmatrix} \sin(\frac{\theta}{2}) * \cos(\beta_x) \\ \sin(\frac{\theta}{2}) * \cos(\beta_y) \\ \sin(\frac{\theta}{2}) * \cos(\beta_z) \\ \cos(\frac{\theta}{2}) \end{bmatrix} = \begin{bmatrix} q1 \\ q2 \\ q3 \\ q0 \end{bmatrix} \tag{C.1}$$

The quaternion is in the form of the vector first (q1,q2,q3) and then the scaler (q0) and θ is the rotation angle and $\cos(\beta_x), \cos(\beta_y)$, and $\cos(\beta_z)$ are the direction cosines locating the axis of rotation.

Quaternions can be easily effected by noise, so having an accurate quaternion relative to your location is important. This system must have low to medium measurement and alignment noise. Currently the system has variable measurement noise. Although the measurement error is low when it is translated into arcseconds it can be seen to grow exponentially. The error is due the cameras' not being able to accurately determine the exact center of the light due to the pixelation of the image. This can be accounted for by dimming the LEDs intensity and making the color deeper. It maybe possible by using different estimation techniques to reduce this measurement error and significantly increase accuracy.

The measurement output from the star tracker is given as Δ. This output is assumed to be given such that Δ provides the inertial coordinates with respect to the star tracker coordinate system, such that:

$$[\ \Delta\ ]_{C}^{Inertial} \longrightarrow \text{Star Tracker Measurement}$$

C.1.4 OmniFlash - Single Board Computer

The OmniFlash single board computer processes all the sensor data to be sent to the math simulation program interface and routes all control signals from the interface to the prototype vehicle. The OmniFlash was chosen due to its adaptability by having a small surface area and all the necessary ports,2 serial ports, 16 digital I/O lines and

Array 1	Array 2	Array 3	Array 4	Array 5 (ceiling)
Red	Green	Blue	Red	Blue
Red	Green	Blue	Red	Blue

Table C.1: Possible light array setups.

Ethernet. This means that we have the ability to increase the number of number of sensors used on the prototype vehicle. Having ethernet ability also allows for wireless transfer of data between our home base and the vehicle, the allows for greater autonomy in the simulations. The OmniFlash comes loaded a 200 Mhz ARM processor, 32 MegaBytes RAM,16 MegaBytes Flash and preloaded with Linux, which makes for ease of programming and fast sensor polling.

One problem encountered is with realtime sensor polling, there is a delay between polling processing and receiving of data. Another problem is it is difficult to maintain connection to the cameras, this may be due to the amount of power that is able to be transferred at any one time over the USB protocol which is a maximum of 500mA, it is possible that the system is activating too many cameras at one time thus overloading the USB demand and closing the COM port. This can be corrected by writing a protocol to make sure that all cameras are off before turning a different camera on.

C.2 Test Environment

The test environment consists of a blacked out encloser. The encloser is 36" x 36" x 36'. Each wall will contain a "colored star pair" arranged in on of the patterns shown below:

Each pattern consists of two colored lights, either the same color or some combination of green, blue, or red. The two long walls will have three sets of two lights the ceiling will have one set of two lights and the short walls will have two sets of two lights. These patterns allow the CASTS too look into it's database to find the appropriate configuration to output the proper quaternion. In order to prevent over lap on set of blue lights are placed on the ceiling of the enclosure and one pair or red lights are placed on opposite walls.

Also inside of the encloser there will be "obstacles". These "obstacles" will be designed as miniature wall, both traversable and not, as well as simulated hills and craters. It is possible that these "obstacles" will both inhibit movement as well as line of sight. This will potentially prove troublesome, but it an integral part of testing. In actual exploration one camera may be faulty or be obstructed and the star tracker CASTS will have to compute the best quaternion with given information.

The enclosure is large enough for the prototype rover to move freely about and around/over obstacles. Due to the limited size of the enclosure though certain values will have to be scaled to work in this small scale test, due to the nature of latitude and longitude. At the equator there is great distance between even 1$\circ$ latitude and longitude, which means long distances must be traversed in order to move small distances according to latitude and longitude. Although at the polar regions this is different. Here there may be only a few meters difference between one line of longitude and the next. Using this assumption we will set up our test enclosure to mimic the polar region of a planetoid.

It was later determined that due to the inconsistency in the ability to calibrate the CASTS system that an inertial navigation system (INS) in the form of an inertial measurement unit (IMU) would be used.

	Mean	STD Dev	Variance
Accelerometer			
x	-0.0876	0.0280	0.0008
y	-0.0678	0.0239	0.0006
z	9.6351	0.0341	0.0012
IMU			
roll	0.2069	0.2099	0.0441
pitch	-0.5497	0.0606	0.0037
yaw	221.4822	0.1265	0.016
magnetic heading	-131.452	0.4319	0.1866
GPS			
Latitude	43.205047	1.181E-5	1.40E-10
Longitude	-70.873453	1.217E-5	1.48E-10

Table C.2: Sensor Statistics [31]

C.3 Inertial Navigation System (INS)

It is determined that an INS/IMU must be chosen that has the ability for consistent calibration. This means that a well documented sensor platform with calibratable sensors is to be chosen. The IMU contains three sensor platforms, a 3-D accelerometer, a 1-axis gyro, a 2-axis gyro, and a magnetometer. On an adjacent sensor platform a GPS is also available. The IMU and GPS is processed through an Arduino this allows for polling of the sensors through a math simulation interface. The following experimental simulations were performed and piloted by Amy Underwood, using the CelNav algorithm and model developed in this document.

The experimental test platform used was a DFRobotShop differential drive rover and controlled by a Anrduino Uno micro-controller[Amy]. The sensors contained in the IMU are as stated above a 3-D accelerometer, a 1-axis gyro, a 2-axis gyro, and a magnetometer as well as an additional GPS shield. [31] The sensor platform is capable of realtime communication with an external wireless communication system. Using this sensor platform the Arduino is able to control the rover's desired heading, thus driving the rover to its final destination.[31] Using this sensor information it is possible to determine all data necessary to use the CelNav algorithm.

One data set that must be generated is the star tracker data. It is not feasible to directly sample star tracker data due to cost, complexity, and light pollution, thus the necessary star tracker data will be determined using one of CelNav's backtracking methods, this specific method is discussed in Chapter 2. Determining star tracker data using this method is possible since the IMU has sensors capable of determining position, GPS, tilt and slope, accelerometer, and heading from the magnetometer, which first needed to be converted into local heading from global.[31] It is important to note that due to the resolution of the mirco-controller significant error introduced into backtracking method which allows for a worst-cast scenario to be tested.[31] Please note this testing is to be a proof of concept for CelNav, and not the final experimental results.

Appendix D

CAST Algorithm

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

%This is the main program!
%It relies on the .m files contained in this folder
%The goal of this program is the determine attitude(yaw, pitch, roll) of
    a
%rover using multiple cameras.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function [angleA]=startracker()
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Global Control Variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num_aquisitions=2;
disk_size=6;%This couldn't find stars at 12


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

% %Red green and blue values
R='Red';
G='Green';
B='Blue';
% D='Blue & Red';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Start of the Initialization Program
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%How to do user replies
message = sprintf('Initialization of Cameras?');
reply = questdlg(message, 'Initialization?', 'Yes','NO', 'Yes');
 if strcmpi(reply, 'Yes')
       % User wants to initialize the system to initial reference points
```

```
   %Get and Image snapshot forward


%%Initialize Each Camera for reference points if reply is YES
% cam_yaxis = videoinput('winvideo',1,'YUY2_1280x960');
% cam_xaxis = videoinput('winvideo',2,'YUY2_1280x960');
% cam_zaxis = videoinput('winvideo',3,'YUY2_1280x960');

%IF it thinks it is an RGB camera
cam_xaxis = videoinput('winvideo',1,'RGB24_1280x960');
cam_yaxis = videoinput('winvideo',2,'RGB24_1280x960');
cam_zaxis = videoinput('winvideo',3,'RGB24_1280x960');

[R1x_xaxis,R2x_xaxis,R1y_xaxis,R2y_xaxis,Color_Ref_xaxis]=initilize(
    cam_xaxis,disk_size);
[R1x_yaxis,R2x_yaxis,R1y_yaxis,R2y_yaxis,Color_Ref_yaxis]=initilize(
    cam_yaxis,disk_size);
[R1x_zaxis,R2x_zaxis,R1y_zaxis,R2y_zaxis,Color_Ref_zaxis]=initilize(
    cam_zaxis,disk_size);

Ref_case_1=strcmp(Color_Ref_xaxis,B) && strcmp(Color_Ref_yaxis,G);
Ref_case_2=strcmp(Color_Ref_xaxis,R) && strcmp(Color_Ref_yaxis,B);
Ref_case_3=strcmp(Color_Ref_xaxis,B) && strcmp(Color_Ref_yaxis,R);
Ref_case_4=strcmp(Color_Ref_xaxis,G) && strcmp(Color_Ref_yaxis,B);

if Ref_case_1==1;
    disp('<==Ref_Case 1==> Blue and Green')
    Ref_case=1;
end



if Ref_case_2==1;
    disp('<==Ref_Case 2==> Red and Blue')
    Ref_case=2;
end

if Ref_case_3==1;
    disp('<==Ref_Case 3==> Blue and Red')
    Ref_case=3;
end

if Ref_case_4==1;
    disp('<==Ref_Case 4==> Green and Blue')
    Ref_case=4;
end
%clear the trigger
flushdata(cam_xaxis)
flushdata(cam_yaxis)
flushdata(cam_zaxis)
%This is where the omni is initialized
init_omni
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Start of the Main Program!!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%Ready to Aquire rotation data?
message = sprintf('Ready to Aquire Rotation Data?');
reply = questdlg(message, 'Ready to Aquire Rotation Data?', 'OK','Cancel
    ', 'OK');
 if strcmpi(reply, 'Cancel')
       % User canceled so exit.
    return;
 end

%  Tell the program what cameras to use when not initializing
% cam_yaxis = videoinput('winvideo',1,'YUY2_1280x960');
% cam_xaxis = videoinput('winvideo',2,'YUY2_1280x960');
% cam_zaxis = videoinput('winvideo',3,'YUY2_1280x960');
%
% % %If it thinks its an RGB camera
cam_xaxis = videoinput('winvideo',1,'RGB24_1280x960');
cam_yaxis = videoinput('winvideo',2,'RGB24_1280x960');
cam_zaxis = videoinput('winvideo',3,'RGB24_1280x960');

%clear the trigger
flushdata(cam_xaxis)
flushdata(cam_yaxis)
flushdata(cam_zaxis)

%This opens the file to write the quaternion measurments
fid = fopen('quat.txt', 'wt'); % was using 'wt'
% fid_ang = fopen('angle.txt', 'wt');
% fprintf(fid_ang, 'The Angles in X,Y,Z\n');
counter=0;
 %This is the loop of the program
 for index=1:num_aquisitions
%This figures out what the angle is given the reference points and a
    camera
disp('Adjust the attitude....Press [Enter] to continue...')
commandwindow;
pause;
% tic
counter=counter+1;

disp('Aquiring Star Locations...')
[Angle_yaxis, Color_xaxis]=angle(cam_xaxis,R1x_xaxis,R2x_xaxis,R1y_xaxis
    ,R2y_xaxis,disk_size);
[Angle_xaxis, Color_yaxis]=angle(cam_yaxis,R1x_yaxis,R2x_yaxis,R1y_yaxis
    ,R2y_yaxis,disk_size);
```

```
[Angle_zaxis, Color_zaxis]=angle(cam_zaxis,R1x_zaxis,R2x_zaxis,R1y_zaxis
    ,R2y_zaxis,disk_size);

if Angle_zaxis==-1 ||Angle_xaxis==-1 || Angle_yaxis==-1
    disp('++++ Could NOT find Stars ++++')
    fprintf(fid, '++++ Could NOT find Stars ++++\n');
else

Angle_Degrees=['X Angle: ' num2str(Angle_xaxis,'%f'),' Y Angle: '
    num2str(Angle_yaxis,'%f'),' Z Angle: ' num2str(Angle_zaxis,'%f')];

disp(Angle_Degrees)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The logic of facing a different direction
%We will add 90 degrees based on what way it is facing!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%strcmp compares two strings and returns true==1 if they are the same.
% case_0=Initialization_Colors
case_1=strcmp(Color_xaxis,B) && strcmp(Color_yaxis,G);
case_2=strcmp(Color_xaxis,R) && strcmp(Color_yaxis,B);
case_3=strcmp(Color_xaxis,B) && strcmp(Color_yaxis,R);
case_4=strcmp(Color_xaxis,G) && strcmp(Color_yaxis,B);

%Angles are in Degrees at this point!

if case_1==1;
    case_actual=1;
    disp('<==Case 1==> Blue and Green')
    Angle_zaxis=Angle_zaxis+90*(abs(case_actual-Ref_case));
end


if case_2==1;
    case_actual=2;
    disp('<==Case 2==> Red and Blue')
    Angle_zaxis=Angle_zaxis+90*(abs(case_actual-Ref_case));
end

if case_3==1;
    case_actual=3;
    disp('<==Case 3==> Blue and Red')
    Angle_zaxis=Angle_zaxis+90*(abs(case_actual-Ref_case));
end

if case_4==1;
    case_actual=4;
    disp('<==Case 4==> Green and Blue')
    Angle_zaxis=Angle_zaxis+90*(abs(case_actual-Ref_case));
```

```
end

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

%  disp(' ====================================================== ');
% disp('Angle_xaxis %f,Angle_yaxis %f,Angle_zaxis %f',Angle_zaxis,
   Angle_xaxis, Angle_yaxis);
%
%  Angle_zaxis
% Angle_xaxis
% Angle_yaxis

Angle_Degrees=['X Angle: ' num2str(Angle_xaxis,'%f'),' Y Angle: '
   num2str(Angle_yaxis,'%f'),' Z Angle: ' num2str(Angle_zaxis,'%f')];
Colors=['X Color: ' num2str(Color_xaxis,'%f'),' Y Color: ' num2str(
   Color_yaxis,'%f'),' Z Color: ' num2str(Color_zaxis,'%f')];
disp(Angle_Degrees)
disp(Colors)

%Convert the angles from DEGREES to RADIANS
Angle_forward=deg2rad(Angle_zaxis);
Angle_right=deg2rad(Angle_xaxis);
Angle_left=deg2rad(Angle_yaxis);


% Stored_Angle(index,1:3)=[Angle_xaxis Angle_yaxis Angle_zaxis];
%Convert from yaw, pitch and roll to a quaternion
Quaternion=angle2quat(Angle_zaxis,Angle_xaxis,Angle_yaxis,'ZXY');
disp('Quaternion:')

disp(Quaternion)
% Stored_Angle(index,1:3)=[Angle_xaxis Angle_yaxis Angle_zaxis];
Stored_Quaternion(index,1:4)=Quaternion;
%Write the quaternion data to the file specified above
fprintf(fid, '%5.4f %5.4f %5.4f %5.4f\n',Quaternion);
disp(' ====================================================== ');
disp(' ====================================================== ');
% Orientation(index,1)=Angle_forward;
% Orientation(index,2)=Angle_right;
% Orientation(index,3)=Angle_left;
%Orientation(index,4,)=Quaternion;

flushdata(cam_zaxis)
flushdata(cam_xaxis)
flushdata(cam_yaxis)
toc
end
 end
%Run the matlab files to command the accelerometer and send the data via
     ftp(ip transfer)
accel_readings(num_aquisitions)
```

```
%Close the file after writing
% fclose(fid_ang);
fclose(fid);
delete(cam_zaxis)
delete(cam_xaxis)
delete(cam_yaxis)
clear cam_zaxis cam_xaxis cam_yaxis
disp(' ===== Collection Complete =====');
%%
load quat.txt
load Accel.csv
disp('The Acceleration readings:')
disp(Accel)
disp('The Startracker readings:')
disp(quat)
% toc
%% Inputs
[rows,cols]=size(Accel);

i=rows;

for i=0:rows,
y = -Accel(i,2);
z = Accel(i,3);
x = Accel(i,4);

fprintf('\n\nStartracker Quaternion\n');
q0 = quat(i,1);
q1 = quat(i,2);
q2 = quat(i,3);
q3 = quat(i,4);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%Definitions
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Define Lunar Coordinate Transformations%%%

   R_moon            = 6371*1e3;
   Omega__SC_I       = eye(3);
   Omega__SD_SC      = eye(3);
   Omega__SD_I       = Omega__SD_SC*Omega__SC_I;
   Q_Omega__SD_I     = A2Q(Omega__SD_I);
   R_Omega__SD_I_E   = eye(3);
   Q_R_Omega__SD_I_E = A2Q(R_Omega__SD_I_E);
   R_Omega__SD_I     = Omega__SD_I;
   Q_R_Omega__SD_I   = A2Q(R_Omega__SD_I);


%%%Alignment%%%
   % Correction Rotation
      U__NED_ENU = [0 1 0; 1 0 0; 0 0 -1];
```

```
    % Body Alignment
       Body       = [1 0 0;0 1 0;0 0 1];

    % Accelerometer Alignment
       Acc        = [1 0 0;0 1 0;0 0 1];

%%%Body with respect to Accelerometer%%%

    % Creates Scalar Portion of Quaternion Vecter
       Body_eta        = cross(Body(3,:),Acc(3,:)');

    % Degrees
       Body_theta      = acos(Body(3,:)*Acc(3,:)')*180*pi;

    % Quaternion Vector Body to Acc z-vector
       Body_quat       = Body_eta*sin(Body_theta/2*pi/180);

    % Quaternion Scalar Body to Acc z-vector
       Body_quat0      = cos(Body_theta/2*pi/180);

    % Total Quaternion
       Q_Acc_1         = [Body_quat(1),Body_quat(2),Body_quat(3),
          Body_quat0]';

    % Normalizes Acc1 Quaternion
       Q_Body          = Q_Acc_1/norm(Q_Acc_1);

    % Rotation Matrix
       R_Body          = Q2A(Q_Body);

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

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


%%%Body with respect to Star Tracker%%%

    % Star tracker z-vector
       neg_Star_tracker_vec    = [0 0 1]';

    % Star tracker z-(unit) vector
       neg_Star_tracker_vec_unit = neg_Star_tracker_vec/norm(
          neg_Star_tracker_vec);

    % Creates Scalar Portion of Quaternion Vecter
       M__C_Body_eta           = cross((Body(3,:)),
          neg_Star_tracker_vec_unit);

    % Degrees
       M__C_Body_theta         = acos(neg_Star_tracker_vec_unit'*Body
       (3,:)');

    % Quaternion Vector Body to Acc z-vector
       M__C_Body_quat          = M__C_Body_eta*sin(M__C_Body_theta/2*pi
          /180);

    % Quaternion Scalar Body to Acc z-vector
       M__C_Body_quat0         = cos(M__C_Body_theta/2*pi/180);

    % Total Quaternion
       Q_star                  = [M__C_Body_quat(1),M__C_Body_quat(2),
          M__C_Body_quat(3),M__C_Body_quat0]';

    % Normalize Star Tracker Alignment Quaternion
       Q_star_n                = Q_star/norm(Q_star);

    % Rotation Matrix
       R_M__C_Body             = Q2A(Q_star_n);

    % Alignment Matrix
       R_Error_Body          = RR_b(0);
       R_M__C_Body_ae          = RR_b(0);

       Q_R_M__C_Body_ae        = A2Q(R_M__C_Body_ae);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%Accelerometer
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%Read in Accelerometer Sensor Data%%%

    % Voltages to Angles
       %x_axis_angle = acc(:,2);
       %y_axis_angle = acc(:,3);
       %z_axis_angle = acc(:,4);
       x_axis_angle = x;
       y_axis_angle = y;
       z_axis_angle = z;

    % Angles to Rotation Matrix

    Gamma_A_Down = [x_axis_angle,0,0;
              0,y_axis_angle,0;
              0,0,z_axis_angle];

    % Rotation Matrix to Quaternion

    Gamma = [x_axis_angle,y_axis_angle,z_axis_angle]';
```

```
%%%Assemble Alpha/Beta/Upsilon Matrix%%%

    %%%Alpha/Beta%%%
        % ST_Down_Body = (Gamma_A_Down*M_Body_A)';

    %%%Upsilon Check%%%
        % Upsilon     = (Omega_Inertial_SD*Delta_C_Inertial*M_Body_C)'*(
            U_NED_ENU *LL_SD_ENU)';




%%%%%%%%%%%%%%%%%%%%%%%%%%%Star Tracker
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Read in Startracker Sensor Data%%%

   % Delta = star;

%%%Assemble Delta Matrix%%%

    %Delta1=[Delta(1);Delta(2);Delta(3)];
    %Delta0=Delta(4);

    Delta1 = [q1;q2;q3];
    Delta0 = [q0];
    Delta_Rot = q2rotmat(Delta1,Delta0);

    Delta__C_Inertial = Delta_Rot;

%%%Assemble Phi Matrix%%%

     Phi__Down_SD_e      =       Gamma_A_Down*...
         % Gamma Rotation Matrix with Noise Addition
                                 R_Error_Body*R_Body*...            %
                                     Accelerometer Alignment and Body
                                     Rotation Matrixes
                                 R_M__C_Body'*R_M__C_Body_ae'*...
                                     % Star Tracker Alignment and Body
                                     Rotation Matrixes
                                 Delta__C_Inertial'*...
                                     % NASA Model
                                 R_Omega__SD_I'*R_Omega__SD_I_E';
                                     % Place Holder for NASA Gravity
                                     Model




%%%%%%%%%%%%%%%%%%%%%%%%%%%Extraction
```

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

%%%Extraction of Lambda/Phi%%%
    Lambda(i) = atan2(-Phi__Down_SD_e(3,2),-Phi__Down_SD_e(3,1))*180/pi;

    Phi(i) = atan2(-Phi__Down_SD_e(3,3),(sqrt(Phi__Down_SD_e(2,3)^2+
       Phi__Down_SD_e(1,3)^2)))*180/pi;

    epsilon(i) = atan2(-Phi__Down_SD_e(2,3),Phi__Down_SD_e(1,3))*180/pi;

%%%Extraction Alpha/Beta%%%

    %alpha = atan2(ST_Down_Body(3,2),ST_Down_Body(2,2));

    %beta = atan2(-ST_Down_Body(1,3),ST_Down_Body(1,1));


%%%Outputs%%%
fprintf('Outputs\n\n');
fprintf('Lambda %3.3f\n\n',Lambda(i));
fprintf('Phi %3.3f\n\n',Phi(i));
fprintf('Epsilon %3.3f\n\n',epsilon(i));
LLH(i,:) =[Lambda(i);Phi(i);epsilon(i)];
end


%% Save
fid = fopen('Locations.csv','w');

for ii = 1:rows,
    fprintf(fid,'%s','Latitude = ');
    fprintf(fid,'%5.2f\n',LLH(ii,1))
    fprintf(fid,'%s','Longitude = ');
    fprintf(fid,'%5.2f\n',LLH(ii,2))
    fprintf(fid,'%s','Heading = ');
    fprintf(fid,'%5.2f\n',LLH(ii,3));
    fprintf(fid,'\n');
end
fclose(fid);
```

Appendix E

Skyscout Recorder

```
% SkyScoutRecorder: records data from a SkyScout for a specified
   quantity of seconds
%  Author         : Charles E. Campbell, Jr., GSFC
%  Based on code by : Mike Lemp, Celestron
%  Date           : Jun 12, 2009
% ======================================================================
% Header Section: {{{1

% ----------------------------------------------------------------------
% Includes: {{{2
#include <iostream>
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
#include <sys/timeb.h>
#include <string>
#include <ctype.h>
#include <sys/select.h>
#include <errno.h>

using namespace std;

% ----------------------------------------------------------------------
% Definitions: {{{2
#define BUFSIZE 256
#define USE_ORIENTATION

% ----------------------------------------------------------------------
% Typedefs: {{{2
typedef int (__stdcall *BulkOpenTYPE)( void );
typedef void (__stdcall *BulkCloseTYPE)( void );
typedef int (__stdcall *versionCmdTYPE)( unsigned short& majorVersion,
   unsigned short& minorVersion, unsigned short& buildVersion );
typedef int (__stdcall *getGPSCmdTYPE)( double& latitude, double&
   longitude, double& elevation, unsigned int& time, int& source, int&
   status );
```

```
typedef int (__stdcall *getLastTargetCmdTYPE)( float& altitude, float&
   azimuth, float& rotation, float& rightAscension, float& declination);
#ifdef USE_ORIENTATION
typedef int (__stdcall *getOrientationCmdTYPE)(float& altitude, float&
   azimuth, float& rotation);
#endif

% ---------------------------------------------------------------------
% Global Variables: {{{2
HINSTANCE           driverDll;
BulkOpenTYPE        BulkOpenAddr;
BulkCloseTYPE       BulkCloseAddr;
versionCmdTYPE      versionCmdAddr;
getGPSCmdTYPE       getGPSCmdAddr;
getLastTargetCmdTYPE getLastTargetAddr;
#ifdef USE_ORIENTATION
getOrientationCmdTYPE getOrientationAddr;
#endif
FILE               *fp              = NULL;
unsigned long       qtysec          = 0L;
char               *outputfile      = NULL;

% ---------------------------------------------------------------------
% Prototypes: {{{2
int  main(int, char **);     %/* SkyScoutRecorder.cpp */
int  SkyScoutInit(void);     %/* SkyScoutRecorder.cpp */
void SkyScoutClose(void);    %/* SkyScoutRecorder.cpp */
int  PrintVersion(FILE *);   %/* SkyScoutRecorder.cpp */
int  PrintSkyScoutData(FILE *); %/* SkyScoutRecorder.cpp */

% =====================================================================
% Functions: {{{1

% ---------------------------------------------------------------------
% main: it all starts here {{{2
int main(
  int   argc,
  char **argv)
{
char          buf[BUFSIZE];
int           markcnt= 0;
int           result;
fd_set        rmask;
struct timeval timespec;

SkyScoutInit(); % load the library

% handle command line
outputfile= ,SkyScout.out,;   % default: <SkyScout.out>
qtysec   = 600;               % default: ten minutes
if(argc > 1) {
        for(--argc, ++argv; argc > 0; --argc, ++argv) {
                if(isdigit(**argv))    sscanf(*argv,'%lu',&qtysec);
                else if(isascii(**argv)) outputfile = *argv;
                }
        }
fp    = fopen(outputfile,'w'); % default: <SkyScout.out>
fprintf(stderr,'SkyScoutRecorder: Will run for %d sec. Output goes to <%
    s>\n',qtysec,outputfile);

% give version
PrintVersion(fp);

% get data from SkyScout once a second for qtysec seconds
while(qtysec--) {
        PrintSkyScoutData(fp);
        // get an input string from the user. non-blocking. one second
            intervals
        FD_ZERO(&rmask);
        FD_SET(STDIN_FILENO,&rmask);
        timespec.tv_sec = 1L;
        timespec.tv_usec = 0L;
        result         = select(STDIN_FILENO+1,&rmask,NULL,NULL,&timespec
            );
        if(result < 0) {
                fprintf(fp,'***error*** select failure#%d\n',errno);
                break;
                }
        if(FD_ISSET(STDIN_FILENO,&rmask)) {
                if(!fgets(buf,BUFSIZE,stdin)) {
                        fprintf(fp,'***error*** fgets indicates end of
                            stdin\n');
                        break;
                        }
                fprintf(fp,'Mark#%-3d: %s',++markcnt,buf);
                }
        }

% close things down
SkyScoutClose();
fclose(fp);

return 0;
}

% ----------------------------------------------------------------------
% SkyScoutInit: this function loads the scoutDriver.dll library {{{2
int SkyScoutInit(void)
{
driverDll = LoadLibrary( 'scoutDriver.dll' );
if(!driverDll) {
        fprintf(stderr,'***error*** unable to open <scoutDriver.dll>\n');
        exit(1);
        }

% get the addresses for dll functions
BulkOpenAddr      = (BulkOpenTYPE)       GetProcAddress( driverDll, '
   BulkOpen' );
BulkCloseAddr     = (BulkCloseTYPE)      GetProcAddress( driverDll, '
   BulkClose' );
versionCmdAddr    = (versionCmdTYPE)     GetProcAddress( driverDll, '
   versionCmd' );
getGPSCmdAddr     = (getGPSCmdTYPE)      GetProcAddress( driverDll, '
   getGPS' );
getLastTargetAddr = (getLastTargetCmdTYPE) GetProcAddress( driverDll, '
   getLastTarget' );
#ifdef USE_ORIENTATION
getOrientationAddr = (getOrientationCmdTYPE) GetProcAddress( driverDll,
   'getOrientation' );
#endif

int retval = BulkOpenAddr(); // open connection to SkyScout
if(retval != 0) fprintf(stderr,'***warning*** problem connecting to
   SkyScout\n');

return retval;
}

% ---------------------------------------------------------------------
% SkyScoutClose: this function closes the SkyScout connection {{{2
void SkyScoutClose(void)
{
BulkCloseAddr();                   % now call close
if(driverDll) FreeLibrary( driverDll ); % free the library
fprintf(stderr,'connection to SkyScout closed\n');
}

% ---------------------------------------------------------------------
% PrintVersion: this function obtains version info from SkyScout and
   prints it {{{2
%   Returns 0=success
%          something else otherwise
int PrintVersion(FILE *fp)
{
int retval= 0;
unsigned short majorVersion;
unsigned short minorVersion;
unsigned short buildVersion;

%call the get version command
retval = versionCmdAddr(majorVersion,minorVersion,buildVersion);

if(retval != 0) {
        fprintf(stderr,'***error** Couldn't get Version information from
            SkyScout');
        SkyScoutClose();
        return retval;
        }

fprintf(fp,'Version: Major#%d Minor#%d Build#%d\n',majorVersion,
    minorVersion,buildVersion);
fflush(fp);
return retval;
}


% -----------------------------------------------------------------------
% PrintSkyScoutData: this function gets time+pose data from SkyScout
    {{{2
%                 Returns 0=success
%                         something else otherwise
int PrintSkyScoutData(FILE *fp)
{
double      elevation;
double      latitude;
double      longitude;
int         retval   = 0;
int         source;
int         status;
unsigned int time;

% get gps info
retval = getGPSCmdAddr( latitude, longitude, elevation, time, source,
    status);
if(retval != 0) {
        fprintf(stderr,'***error*** can't get GPS information from
            SkyScout');
        SkyScoutClose();
        return retval;
        }

fprintf(fp,'latitude=%f longitude=%f elevation=%f time=%d source=%d
    status=%d ',
  latitude,longitude,elevation,time,source,status);

% Get last target data
float altitude;
float azimuth;
float rotation;
float rightAscension;
float declination;

retval = getLastTargetAddr( altitude, azimuth, rotation, rightAscension,
     declination);
```

```
if(retval != 0) {
        fprintf(stderr,'***error*** can't get last-target information
            from SkyScout');
        SkyScoutClose();
        return retval;
        }

#ifdef USE_ORIENTATION
retval = getOrientationAddr(altitude,azimuth,rotation);
if(retval != 0) {
        fprintf(stderr,'***error*** can't get orientation information
            from SkyScout');
        SkyScoutClose();
        return retval;
        }
#endif

fprintf(fp,''altitude=%f azimuth=%f rotation=%f rightAscension=%f
    declination=%f\n',
  altitude, azimuth, rotation, rightAscension, declination);

fflush(fp);

return retval;
}

% ----------------------------------------------------------------------
% Modelines: {{{1
% vim: fdm=marker
```

Appendix F

Euler Angle Rotation Matrices (Direction Cosine Matrix)

all in order of ϕ, θ, ψ

Roll - Pitch - Yaw (123)

$$\begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta) \\ \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

RPR (121)

$$\begin{bmatrix} \cos(\theta) & \sin(\theta)\sin(\psi) & -\sin(\theta)\cos(\psi) \\ \sin(\phi)\sin(\theta) & -\sin(\phi)\cos(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta)\cos(\psi) + \cos(\phi)\sin(\psi) \\ \cos(\phi)\sin(\theta) & -\cos(\phi)\cos(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) \end{bmatrix}$$

RYR (131)

$$\begin{bmatrix} \cos(\psi) & \sin(\psi)\cos(\psi) & (\sin(\psi))^2 \\ -\cos(\phi)\sin(\psi) & \cos(\phi)(\cos(\psi))^2 - \sin(\phi)\sin(\psi) & \cos(\phi)\cos(\psi)\sin(\psi) + \sin(\phi)\cos(\psi) \\ \sin(\phi)\sin(\psi) & -\sin(\phi)(\cos(\psi))^2 - \cos(\phi)\sin(\psi) & -\sin(\phi)\cos(\psi)\sin(\psi) + \cos(\phi)\cos(\psi) \end{bmatrix}$$

RYP (132)

$$\begin{bmatrix} \cos(\theta)\cos(\psi) & \sin(\theta) & -\cos(\theta)\sin(\psi) \\ -\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) & \cos(\phi)\cos(\theta) & \cos(\phi)\sin(\theta)\sin(\psi) + \sin(\phi)\cos(\psi) \\ \sin(\phi)\sin(\theta)\cos(\psi) + \cos(\phi)\sin(\psi) & -\sin(\phi)\cos(\theta) & -\sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) \end{bmatrix}$$

PYR (231)

$$\begin{bmatrix} \cos(\phi)\cos(\theta) & \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) \\ \sin(\phi)\cos(\theta) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) \end{bmatrix}$$

PYP (232)

$$\begin{bmatrix} \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta) & -\cos(\phi)\cos(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta)\cos(\psi) & \cos(\theta) & \sin(\theta)\sin(\psi) \\ \sin(\phi)\cos(\theta)\cos(\psi) + \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta) & -\sin(\phi)\cos(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) \end{bmatrix}$$

PRP (212)

$$\begin{bmatrix} (\cos(\phi))^2 - (\sin(\phi))^2\cos(\theta) & \sin(\phi)\sin(\theta) & -\cos(\phi)\sin(\phi) - \sin(\phi)\cos(\theta)\cos(\phi) \\ \sin(\phi)\sin(\theta) & \cos(\theta) & \cos(\phi)\sin(\theta) \\ \cos(\phi)\sin(\phi) + \sin(\phi)\cos(\theta)\cos(\phi) & -\cos(\phi)\sin(\theta) & -(\sin(\phi))^2 + (\cos(\phi))^2\cos(\theta) \end{bmatrix}$$

PRY (213)

$$\begin{bmatrix} \cos(\phi)\cos(\psi) - \sin(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\theta) \\ -\cos(\theta)\sin(\psi) & \cos(\theta)\cos(\psi) & \sin(\theta) \\ \sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\sin(\psi) - \cos(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

YRP (312)

$$\begin{bmatrix} \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) & -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) \\ -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & -\sin(\theta) & \cos(\theta)\cos(\psi) \end{bmatrix}$$

YRY (313)

$$\begin{bmatrix} -\sin(\phi)\cos(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\cos(\theta)\cos(\psi) + \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\theta) \\ -\cos(\phi)\cos(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) & \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\theta) \\ \sin(\theta)\sin(\psi) & -\sin(\theta)\cos(\psi) & \cos(\theta) \end{bmatrix}$$

YPY (323)

$$\begin{bmatrix} \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) & \cos(\phi)\cos(\theta)\sin(\psi) + \sin(\phi)\cos(\psi) & -\cos(\phi)\sin(\theta) \\ -\sin(\phi)\cos(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & -\sin(\phi)\cos(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \sin(\phi)\sin(\theta) \\ \sin(\theta)\cos(\psi) & \sin(\theta)\sin(\psi) & \cos(\theta) \end{bmatrix}$$

YPR (321)

$$\begin{bmatrix} \cos(\phi)\cos(\theta) & \sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\sin(\psi) - \cos(\phi)\sin(\theta)\cos(\psi) \\ -\sin(\phi)\cos(\theta) & \cos(\phi)\cos(\psi) - \sin(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) \\ \sin(\theta) & -\cos(\theta)\sin(\psi) & \cos(\theta)\cos(\psi) \end{bmatrix}$$