DEVELOPMENT OF THE NASA CELESTIAL NAVIGATION METHOD FOR DYNAMIC EXTRA-TERRESTRIAL SURFACE NAVIGATION

BY

© Jared Perkins

B.S., University of New Hampshire, 2012

THESIS

Submitted to the University of New Hampshire

in partial fulfilment of

the requirements for the degree of

Master of Science

in

Mechanical Engineering

May 2011

This thesis had been examined and approved.

Thesis Director, Dr. May-Win L. Thein

Associate Professor of Mechanical Engineering

Dr. Barry Fussel

Professor of Mechanical Engineering

Dr. L. Gordon Kraft, III

Professor of Electrical and Computer Engineering

Date

Acknowledgements

I want to give my great thanks to all those who have been supportive these last eight years. 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 Prof. May-Win Thein for giving me the the opportunity to work with NASA as a Research Assistant and as a UNH teaching assistant.

I want to thank all at UNH and all my friend and family for all there support, and for a great experience at U-N-H 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 EXTRA-TERRESTRIAL SURFACE NAVIGATION

by

Jared Perkins

University of New Hampshire, July 2011

The Celestial Navigation (CelNav) method was developed in conjunction with NASA Goddard Space Flight Center, to provide accurate location data without using a global positioning system (GPS) or a ground/relay station for extraterrestrial surface navigation. CelNav is a minimal sensor/power solution originally developed for dynamic 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 extra-terrestrial surface navigation when accurate location coordinates are necessary. Preliminary results show that this research shows promise as a secondary dynamic navigation system for future extra-terrestrial exploration.

Introduction

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 or communication with ground stations).

This basic idea was proposed in 2006 by David Quinn. He deducted that with the rudimentary instruments already contained within space vehicles it would be possible to accurately determine one's location, if the error was small. 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 May-Win Thein, and presented in Thein et al. Using this information CelNav was further refined and streamlined into the algorithm that currently stands.

CelNav History

CelNav was originally designed to be used at the Lunar poles in search of pockets of frozen water that have been previously 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.

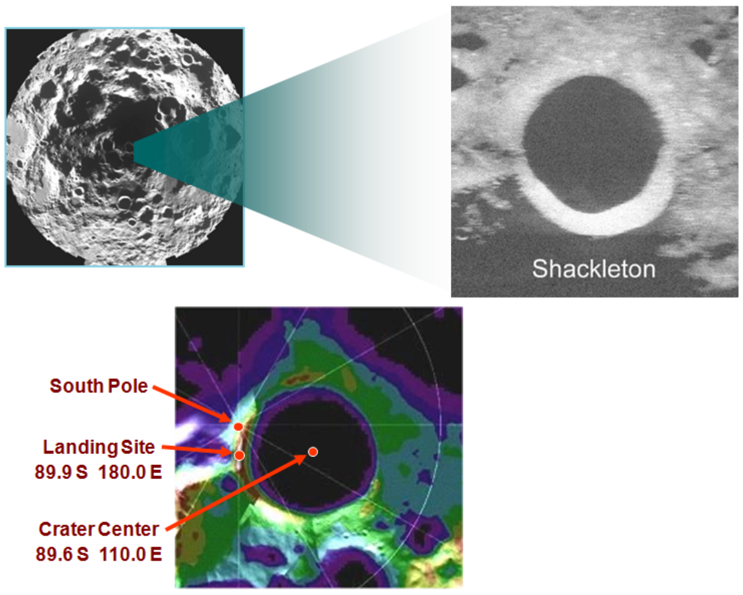

Figure 1: Shackelton Crater located on Earth's Moon

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 (sextant, astrolabe, octant, 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 (Almanac). The Nautical Almanac is based on the fact that any heavenly body has a distinct location above the Earth's surface during certain times of the year. This is known as the heavenly body's geographic position (GP) as found in the Nautical Almanac (Almanac).

There are known relations between the angle of the celestial body and the visible horizon, and this information has a direct relation between GP and the observer's position. From these relations the observer is able to draw a line of position (LOP), and place it on a navigational chart. The observer's position is located somewhere 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" (Almanac).

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 body's. Of these tools, the sextant 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 is that it relies heavily on an accurate chronometer, set to a clock on the meridian (Ifland).

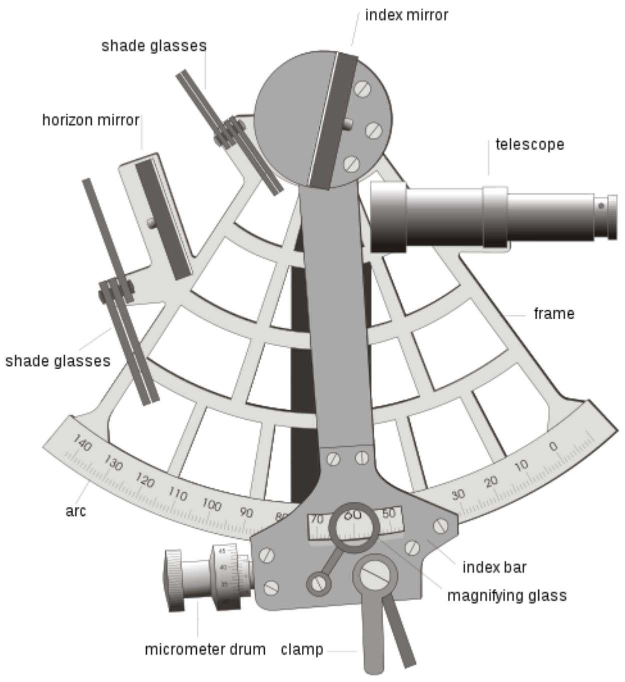

Figure 2: Marine Sextant [Gaspar]

CelNav was conceived out of a need for a reliable inertial navigation system (INS) that can be run independently from outside sources, including, but not limited too, an orbiting Global Positioning System (GPS) type system or orbiting satellite(s).

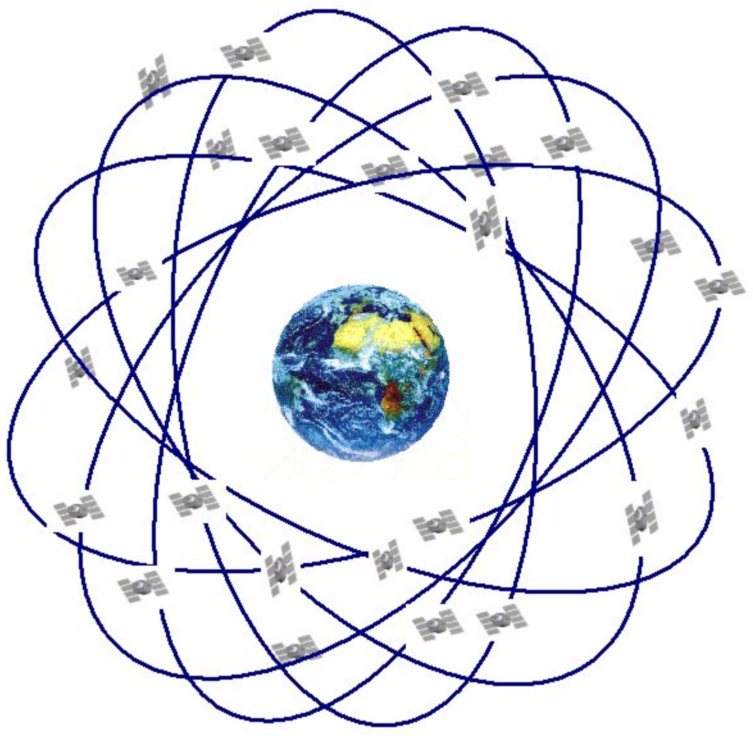

Figure 3: Global Positioning System [Brown]

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

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

Using this information from a minimum of three satellites it is possible to obtain the latitude and longitude coordinate. GPS satellites allow for the altitude measurements. 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 age of satellite hardware.

For the above reasons, as well as many more not listed, there is an impracticality of installing GPS type systems about planetoids, (e,g. ,planet, moon, comet, or meter) being explored/surveyed.

Another method that could be used for navigation on extraterrestrial bodies involves sending, at a minimum, a single satellite to orbit the body. This would allow 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.(Corum) 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 (Corum). The Doppler Effect is useful due to that fact that the orbital information (altitude, velocity, and orbit) would be well known and carefully monitored. Thus, a receiver on a rover or astronaut would be able to tell navigational information based upon the signal received from the orbiting satellite. This navigation method too, however, is prone to many shortfalls, such as: line of sight, limited fuel/power, and solar activity.

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

In the future, long term extraterrestrial missions are expected to have limited power and sensor resources available to both astronauts and rovers. This means that although redundant sensors could be incorporated, but there would be no access to additional sensor measurements. Only sensors crucial to navigation would be available. 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 systems also make it possible to use CelNav to fault check the position data against other known 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 will be explained in greater detail in a later chapter). 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.

3-D accelerometers, also called an accelerometer triad, are three orthogonally arranged accelerometers. Orthogonality is used to describe orientation each axis of the accelerometer falls on the x-y-z axis of the astronaut or rover.

On Board Guidance, Navigation, and Control

CelNav function is to act as a "stand alone" algorithm method 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 due to 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 ground station is a great distance from the rover or astronaut. This is due to 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 on 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:

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

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

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.

Past CelNav Development

The original CelNav problem statement as stated in Thein et al. referred to the Lunar surface. A local coordinate system for the Lunar surface is pre-defined as the East-North-Up (ENU) orthogonal triad. It is 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 the rover was stationary.

Future exploration to extraterrestrial bodies will most likely include surveying of land, soil, minerals and especially water. Future space settlements would depend on this liquid resource. 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 the minimal use of sensors: one 3-D accelerometer and one star tracker.

CelNav originally needed to be accurate within 50 meters (m) of the desired target location. In order to achieve this goal it was determined that that 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. Later it was determined that on the Lunar surface, where a rover or astronaut could be able to see about 1000m away and, therefore, would be an acceptable accuracy target for emergency navigation.

Navigational Techniques

A great deal of current research focuses on extraterrestrial navigation by means of knowing exact ephemeris data, such as an Aeronautical Almanac. As stated in Malay et al., 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 position within

100m. This almanac is limited, though, and needs to be updated on a regular basis. 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 charts becomes completely inaccurate after one year. CelNav is a more robust method then the one to follow 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, "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 with greater knowledge of the ephemeris data the accuracy can be further improved. This method is useful but limited to extra terrestrial bodies with closely orbiting bodies.

Another such navigation method for determining latitude and longitude can be found in Swanzey et. al. This method is similar to CelNav but requires the development of a new sensor, which takes aspects many of the peripheral sensors already included on a navigation platform. A downfall of this method is the requirement of a new sensor package rather than using the sensors that are potentially already available to an astronaut or a rover.

Scope of Research

The current focus is developing a dynamic navigation system that will focus on the need for autonomy. In order for autonomous control, estimators are needed for control feedback. Due to the need for multiple filter/estimations techniques will be examined, to be discussed later. No controller will be examined in this thesis as it is out of the current scope of this research.

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 four 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

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.

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. The model is taken from Sun et al. 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 "measuerments" 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.

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 App F.

The experimental system is used to do a comparative analysis between Sliding Mode Observer (SMO), Extended Kalman Filter (EKF) and H-Infinty (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.

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, Overview of estimation techniques including Extended Kalman Filter, H-Infinity and Sliding Mode Observer.
- Chapter 8, Overview of analytical results the design and result of the estimators (EKF, H_∞ and SMO).
- Chapter 9, Overview of a physical experiment and how it simulates the actual rover/astronaut environment using only two available sensor, accelerometer and star tracker.
- Chapter 10, Experimental results of the estimators (EKF, H_∞ and SMO) implemented on the UNH Lunacat Lunabot.
- Chapter 11, Discussion focusing on relating the three estimation techniques as well as listing pros and cons about using each type for the task of CelNav; also comparing the experimental results to the analytical.
- Chapter 12, Summary and comparison of results of all estimation techniques. Also possible future direction this project can take.

Chapter I

Coordinate Reference Frames

To accurately determine a vehicle's/astronaut's current position, using CelNav as part of a mobility solution, it is necessary to determine the vehicle coordinate (i.e. body) frame with respect to the global (inertial) coordinate frame. Coordinate transformation is required

1 Local and Planetary Reference Frames

The two major coordinate systems for local and global reference frames are Moon Centered Moon Fixed (MCMF) and East-North-Up (ENU).

The coordinate systems in this research are defined such that:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \longrightarrow \text{Body Coordinates System}$$

$$\begin{bmatrix} I \\ II \\ III \end{bmatrix} \longrightarrow \text{Selenodetic Coordinates Systems}$$

$$\begin{bmatrix} E \\ N \\ U \end{bmatrix} \longrightarrow \text{Local Surface Coordinates}$$

As shown in Figure 1.1 through Figure 1.2, each coordinate frame is fixed at a

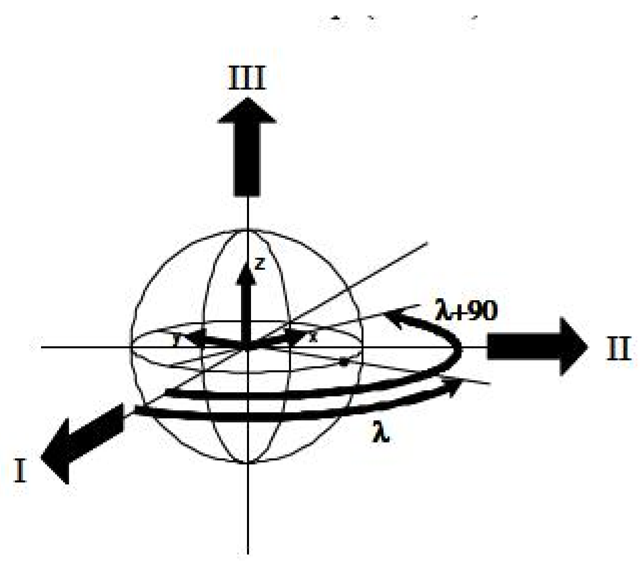

Figure 1.1: Moon Centered Moon Fixed Reference Frame shown (I-II-III triad)

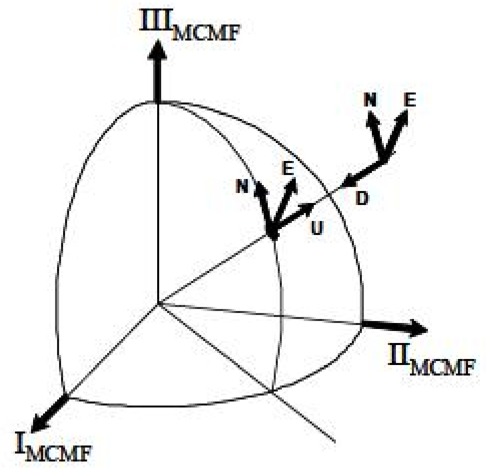

Figure 1.2: ENU Reference Frame

different origin with respect to the planetary lunar body.

A "Down" reference frame will also be used. "Down" can be conveniently used to represent [E, N, U] coordinate system as the "negative Up" position and with a generic heading ϵ (i.e., as in the resulting output of an accelerometer). The ENU reference frame is used due to the fact that aviators commonly use this coordinate reference frame in navigation.

2 Attitude Review - Euler Angles and Quaternions

There are two well known methods for defining the attitude (orientation) of an object in three dimensional space: Euler angles and quaternions. Both systems each have their advantages and disadvantages and will be discussed in the following section.

2.1 Euler Angles

Euler angles are a way to describe any rotation in a minimum of three separate rotations. There are three common Euler angle representations: ϕ, θ, ψ.

A theorem by Euler states: - Any two independent orthonormal 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.[Ben]

These rotations can be described as:

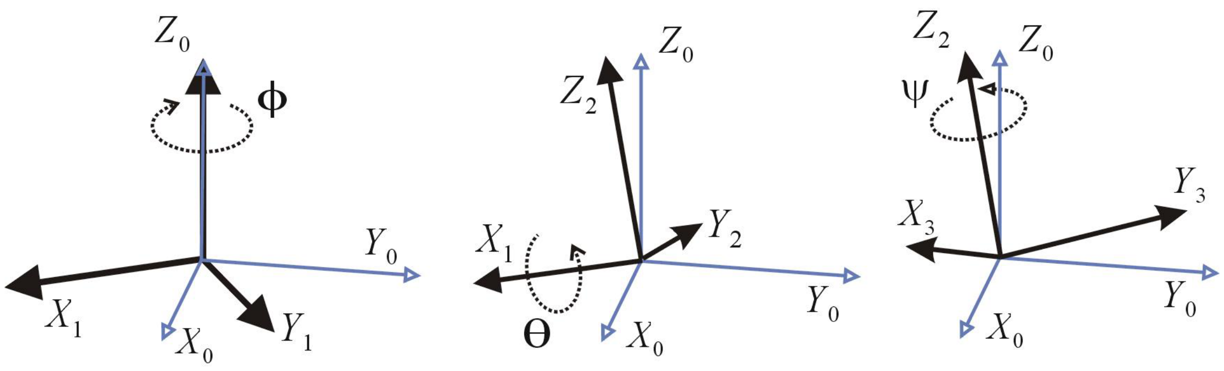

Figure 1.3: Euler Angle Rotations

- Rotation 1 is ϕ rotated about the z-axis.
- Rotation 2 is θ rotated about the former x-axis.
- Rotation 3 is ψ rotated about the former z-axis.

The rotation seen above in Fig. 1.3 is known as a 3-1-3 rotation which is just one of many combinations of possible rotation, a list can be found in Appendix E.

2.2 Direction Cosine Matrix

The most easily understood way to display Euler angles is through the use of a Direction Cosine Matrix (DCM). A DCM is a way combining any three Euler angle rotations [Ben]. This is done thru the multiplication of the three rotation matrices. The rotation matrices that govern a general 3-2-1 rotation can be written as:

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

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

$$\text{Rotation } 2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{1.3}$$

A complete listing of all DCM Euler angles can be found in Appendix E [Ben].

Using Euler Angles has its advantages, as they are more easily understood when describing “Roll”,“Pitch”, and “Yaw” Fig 1.4.

One disadvantage, is that there is the possibility for potential singularities in the system. A gimbal is a platform or ring that is free to rotate about one axis. In the case of aeronautics there are three gimbals as referenced in Fig 1.5.

This configuration of the gimbals can be used as an Inertial Measurement Unit (IMU). Thus, by mounting the set of gimbals to the center of the local system it is possible to obtain the system orientation. The problem occurs when any two gimbals line up, this is called gimbal lock, thus reducing the movement of the gimbals to less than three degrees of freedom (Hoag). A mathematical equivalent of this gimbal lock

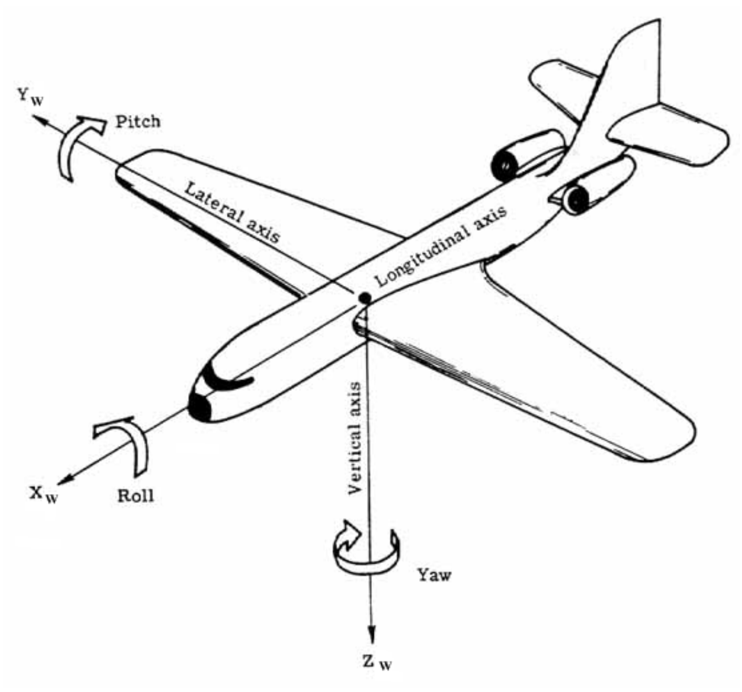

Figure 1.4: "Roll","Pitch", and "Yaw"

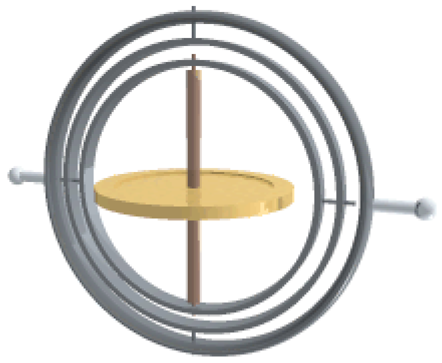

Figure 1.5: Gimbal Lock

may also be shown e.g. 3-1-3 rotation sequence and θ=0; resulting 3-1-3 attitude matrix only have two rotations. This can be solved in a number of ways. One such way is to add another gimbal. Another is to physically move one of the locked gimbals to free the motion. Another way, the one that is focused on in this research, is to use quaternions instead of euler angle.

2.3 Quaternions

The rotations described above can easily be represented in quaternions. The basic form a quaternion takes can be seen in Eq 1.4.

$$q = \begin{bmatrix} q_0 \\ iq_1 \\ jq_2 \\ kq_3 \end{bmatrix} \tag{1.4}$$

or,

$$q = \begin{bmatrix} q_0 \\ q \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\frac{\Phi}{2}) \\ e_1 \sin(\frac{\Phi}{2}) \\ e_2 \sin(\frac{\Phi}{2}) \\ e_3 \sin(\frac{\Phi}{2}) \end{bmatrix} \tag{1.5}$$

First we will look at what a quaternion is and then break it down into its parts. A quaternion is made of two distinct parts, a scalar,consisting of q0, and a vector, consisting of (q1,q2,and q3). Also show in Eq 1.5 is the rotation axis e and the angle Φ. This is best shown by the basic equations for quaternion algebra:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{1.6}$$

This form is similar to:

$$q0^2 + q1^2 + q2^2 + q3^2 = 1 \tag{1.7}$$

This shows a unit quaternion can also be normalized. This proof is necessary due to that fact 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 looks like a fluid rotation, where as Euler rotations are broken up into multiple separate rotations.

It is possible to write an attitude matrix similar to the Euler Angle attitude matrix such that:

$$A = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2 \\ 2(q_1q_2 - q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2 & 2(q_2q_3 - q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{1.8}$$

As shown above the major difference between the Euler angle attitude matrix and the quaternion attitude matrix is the quaternion version does not have any triginometric functions. It is important to note that both the Euler angle rotation matrix and the quaternion rotation matrix will give you the same result for the same location. Thus, there is an inate advantage to using the less computationally intensive method for determioning a rotation matrix. [Kerem]

Both of the above methods were used in the development of CelNav and in the various estimation techniques that will be described in the following chapters.

Chapter II

Celestial Navigation

1 Assumptions

The first major assumption that is made is the determination of the $[\Omega]_{SD}^{Inerital}$. 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) is that almost all to the sensor error comes from the misalignment of the accelerometer.

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. 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 (Thein et al.).

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.

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 is 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 accelerometer with respect to coordinate frame Body (vehicle/astronaut). That is:

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

Likewise,

$$[A] = [M_A]^A_{Body}[Body]$$

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

$$[C] = [M_A]^C_{Body}[Body]$$

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 (Curtin). 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), which 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.

3.1 Accelerometer Model

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

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

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 E:

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

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

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

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

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

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

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)

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

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

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

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

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:

$$\Gamma_{Total} = \begin{bmatrix} \Gamma 1 & \Gamma 2 & \Gamma 3 \end{bmatrix} \tag{2.18}$$

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

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.

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:

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

where,

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

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

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

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 convention to the North-East-Down convention used in this paper. The matrix can be written as follows:

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

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.

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

4 Extracting Tile, Slope, Heading, Longitude, and Latitude Given Accelerometer and Star Tracker Data

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

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

[LL(λ, ϕ)]$_{SD}^{NED}$ 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 Eqn.(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 Eqn. 2.34 that it would be also possible to extract ϕ as seen:

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

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

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:

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

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.

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

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

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

It was previously stated that:

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

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

$$= \left(\begin{bmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) & \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) & -\sin(\beta)\cos(\alpha) \\ -\cos(\alpha)\sin(\epsilon) & \cos(\alpha)\cos(\epsilon) & -\sin(\alpha) \\ \sin(\beta)\cos(\epsilon) + \cos(\beta)\sin(\alpha)\sin(\epsilon) & \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) & \cos(\beta)\cos(\alpha) \end{bmatrix} \right)^T$$

Taking the transpose yields:

$$= \begin{bmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) & -\cos(\alpha)\sin(\epsilon) & -\sin(\beta)\cos(\alpha) \\ \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) & \cos(\alpha)\cos(\epsilon) & \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) \\ -\sin(\beta)\cos(\alpha) & \sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{2.44}$$

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

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

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

$$\epsilon = \tan^{-1}\left[\frac{-\Psi_(1,2)}{\Psi(2,2)}\right] \tag{2.45}$$

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:

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

$$= \begin{bmatrix} \cos(\beta) & \sin(\beta)\sin(\alpha) & -\sin(\beta)\cos(\alpha) \\ 0 & \cos(\alpha) & \sin(\alpha) \\ \sin(\beta) & -\cos(\beta)\sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{2.46}$$

where,

$$[ST(\alpha, \beta)]^{Body}_{Down} = ([\Gamma^{Down}_{Acc}][M_A]^{Acc}_{Body})^T \tag{2.47}$$

such that

$$[\Gamma, M_A]^{Down}_{Body} = [\Gamma^{Down}_{Acc}][M_A]^{Acc}_{Body} \tag{2.48}$$

By combining Eqn. (2.46), (2.47) and (2.48), it is possible to directly extract α and β:

$$\begin{aligned} \alpha = & \tan^{-1}\left[\frac{[\Gamma, M_A](3,2)}{[\Gamma, M_A](2,2)}\right] \\ \beta = & \tan^{-1}\left[\frac{-[\Gamma, M_A](1,3)}{[\Gamma, M_A](1,1)}\right] \end{aligned} \tag{2.49}$$

Chapter III

Errors and Disturbances

CelNav was developed as a cheap and effective mobility solution for the navigation of extra planetary bodies. Lumped error analysis was shown to by Thein et. al. to achieve a 50m accuracy with a maximum of 5.93 arcsec of total noise throughout the entire system. The updated problem statement would allow for errors upwards of 1500m, this can be achieved even with the maximum expected noise levels in the system.

CelNav works as a two fold system, both 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.

By using the appropriate matrices it is possible to extract all necessary navigational data from the CelNav algorithm using accurate Γ and Δ matrices.

CelNav algorithm works as an effective solution, and possible fault detection algorithm, to extract and validate latitude, longitude, heading, tilt, and slope given both a Γ and Δ matrices.

This chapter will focus on the errors and disturbances associated with and expected from the CelNav system. Also included are both measurement and alignment errors, both from a "real" star tracker (Contained Attitude Star Tracking Sensor, (CASTS) and a 3D accelerometer), as well as any errors associated with necessary initial conditions.

CASTS will be discussed in greater detail relating to structure, design and look in Chapter 9.

1 Accelerometer Errors

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

1.1 Accelerometer Measurement Error

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) 3D-accelerometer is needed to generate a unique matrix denoted by Φ (Eqn. 2.32). From this unique Φ, latitude and longitude is extracted; a Γ matrix of the form found in Eqn. 2.19 is also needed. The expected total error of the 3D-accelerometers is expected to be on the order of 10^{-3}.

1.2 Accelerometer Alignment Error

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, the accelerometer requires a maximum alignment error be limited to 60 arcsec. This includes the internal orientation of the 3D accelerometer as well as the mounting of the hardware to a rover or astronaut.

This mounting error accounts for the most significant effect on the accuracy of 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 Fig 3.1.

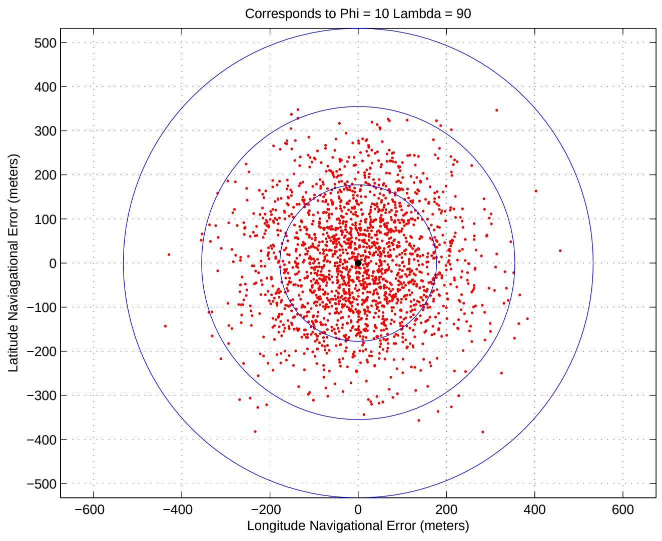

Figure 3.1: 10-20 Arcsec Accelerometer Alignment Noise

2 Simulated Star Tracker and Contained Attitude Star Tracking Sensor (CASTS)

This section focuses on both the measurement and alignment errors found in the star tracker Contained Attitude Star Tracking Sensor (CASTS).

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 will be necessary, with one angled towards the front of rover/astronautn and one angled to the rear. This is necessary to account for instances when one camera may have its field of view blocked by terrain.

2.1.1 Simulation Measurement Error

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 engineers. 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.

2.1.2 CASTS Measurement Error

CASTS measurement error is highly dependent on proper calibration. This is due to one of the colors being repeated around the outside, as see in Fig 3.2.

Figure 3.2: CASTS Experimental Star Setup

Another difficulty to note in the CASTS experimental setup is making sure the colored stars are of the appropriate brightness and not washed out by ambient light or being overly bright. Again it is important to note that if the calibration is done level without any tilt in the "star field" samples will be erratic, but if the calibration is done with a moderate tilt in the "star field" samples will be less erratic. Such that if the first result could be 70deg north latitude the second could be 45deg north latitude with level calibration, but with the moderate tilt the first result could be 70deg north latitude the second could be 70.5deg north latitude without moving the CASTS experimental setup.

2.2 Alignment Error

In this section two different types of alignment error are discussed. First is the star tracker alignment error that is used in the analytical simulation. Next is the alignment error found in the CASTS's sensor.

2.2.1 Simulation Alignment Error

The star tracker alignment matrix is constructed to have a max of 60 arcsec of 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.

2.2.2 CASTS Alignment Error

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

Figure 3.3: CASTS Experimental Star Setup

As can be seen in Fig 3.4. placement of the cameras inside the must be consistent, to help facilitate consistency velcro is added to the bottom of the "star cameras" so the cameras could only be attached one location on the sensor platform. If the "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 Fig 3.4.

Figure 3.4: CASTS Star Camera Viewing Angle

Again placing the “star cameras” consistently is the same location will produce repeatable results.

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, in latitude and longitude, is known. This is important due to implementation of dynamic observers to improve the accuracy of CelNav results. These observers will be discussed at length in Chapter 7. A priori data for latitude and longitude can be extracted from 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.

4 Inertial Navigation System (INS)

It was found through many rounds of testing of the CASTS experiment that it was too difficult to return consistent results due to the setup of the current iteration of the CASTS experiment. Due to the inconsistency in the ability to calibrate CASTS it was determined that an Inertial Measurement Unit (IMU) containing a GPS, 3-D Magnetometer, and 3-D Accelerometer would be used in place of a CASTS experimental setup. This will be discussed in further detail in a later chapter.

Chapter IV

CelNav Performance Analysis

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. 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 is if the answer is known and it is necessary to find out how close an experiment is to providing that answer; Monte Carlos Analysis develops a range of possible answers with a more accurate solution that is obtainable by running more experiments. 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 sequences of states whole evolutions is determined by random events." (Kalos) 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.

1.1 Testing Parameters

This testing will be looking at a "global map" of the lunar surface specifically looking at major latitudes and longitude moving to minor latitudes and longitude approaching the polar regions. This chapter will look at navigational error associated with both normal and worst case noise levels and how Monte Carlo parameters are calculated.

2 Monte Carlo Parameters

For this set of Monte Carlo simulations the following 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. These parameters will be discussed at length in the following sections.

2.1 Sample Size

The number of required sample points is dependent upon the predicted sampling error and some bounding term B such that (Kalos):

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

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, the above equation shows that there must be between ≈1000 and 1600 samples before a large enough random sample pool is available. The table below shows the number of sample points that CelNav is able to produce in a single simulation period, 2028, due to how how the analytical CelNav algorithm is setup. Compared to 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 noise that can be described as white noise.

2.2 Locations to Sample

One must also to choose which latitudes and longitudes to sample. It is important to note that for this Monte Carlo analysis, testing will encompass the entire lunar surface in latitude and longitude. This is important for repeatability of certain samples. Repeatability is necessary due to the fact that the simulation should have the same sample distribution for a constant latitude while rotating the body about different longitudes. This should hold true for regions with large distances between both latitudes and longitudes, which can be found in the table below:

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

3 Uncertainties and Biases

Choosing appropriate noise levels is the next step. The noise levels are the parameters that are to be varied in the Monte Carlo simulations. 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 noise that could be expected in the potential 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 Levels

4 Data Distribution

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

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

As can be seen in Table 4.4:

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

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

The mean and standard deviation for the given data set shown in Fig 4.1.

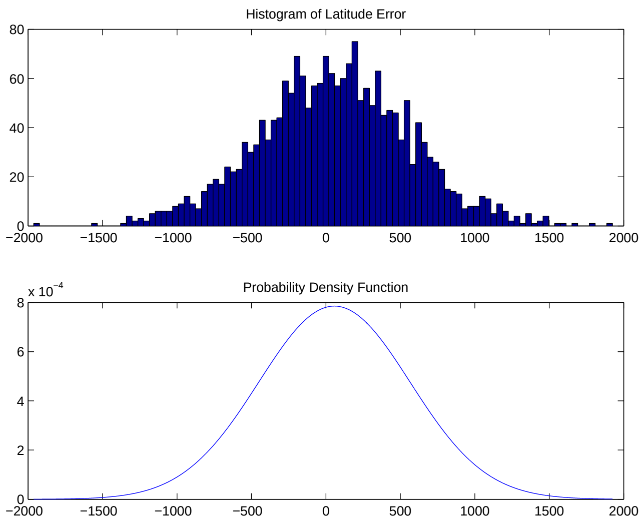

Figure 4.1: Probability Distribution Function - CelNav Simulation

Showing the above is necessary for the development of the Extended Kalman Filter (EKF), to be discussed in later chapters.

5 Results

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

Three different Lunar regions will be explored including the equator and the polar regions (North Pole/South Pole).

In each of the following figures, three circles are drawn which coincide with various confidence intervals, the x-axis shows latitude navigational error and the y-axis show longitude navigational error, both in meters. The latitude navigational error can be calculated as:

$$\phi\text{naverr} = \frac{\phi_e}{180} * R_{moon} * pi \tag{4.4}$$

Also the longitude navigational error can be calculated as:

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

The confidence intervals moving from smallest to largest can be labeled as the following:

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

The table below shows what criterion were 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

5.1 Test Case - Latitude 0 and Longitude 90

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

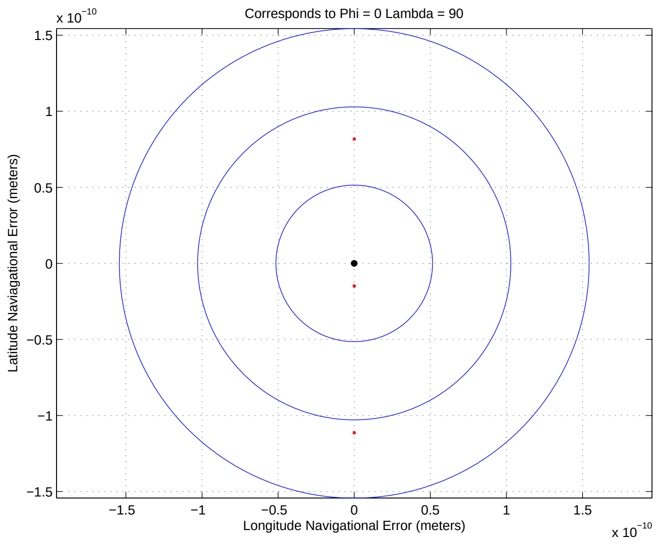

Figure 4.2: Case 1 (No Noise) - Latitude 0° Longitude 90°

Next only accelerometer measurement noise for the "expect noise" simulation will be discussed. It can be seen in Fig 4.3 that there is more scattering of locations as compared to Fig 4.2.

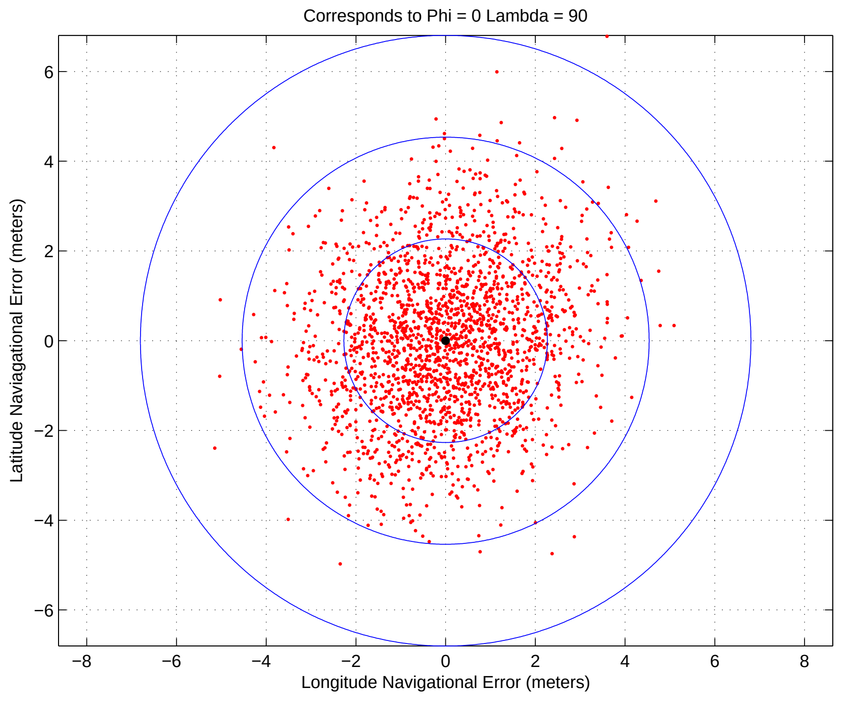

Figure 4.3: Case 2 (Accelerometer Measurement Noise) - Latitude 0° Longitude 90°

As it can be seen above in Fig 4.3 with only marginal measurement noise almost all of the sample data points fall within a 4m radius 2-σ circle.

Below it can be seen that with worse case noise the distance error increases to 2000m. This is a great increase in distance, but if used by an astronaut it is an acceptable distance, due to the fact that it is possible to clearly see upwards of 2500m on the Lunar surface (Quinn). As long as a majority of the possible locations sampled appear within the 3-σ circle, the error thought large, is still a valid solution (Quinn).

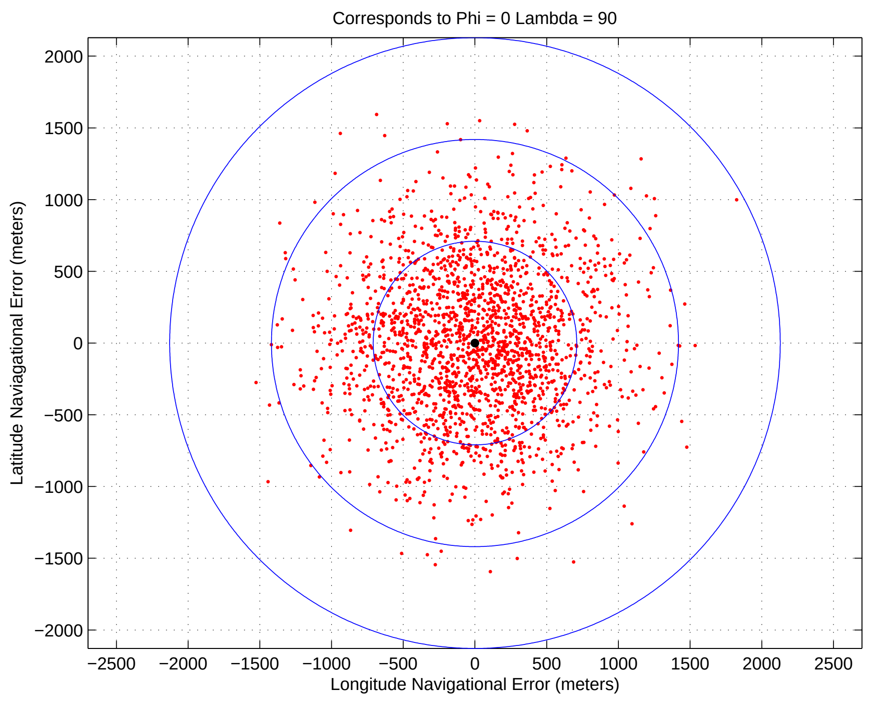

Figure 4.4: Case 3 (Worst Case Noise) - 0° Longitude 90°

5.2 Test Case - Latitude 89.9 Longitude 90

The next set of test cases are as follows:

	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 Fig 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 Fig 4.2 meaning that when instrumentation is even extremely close to the polar regions it is still possible to get acceptable navigational results.

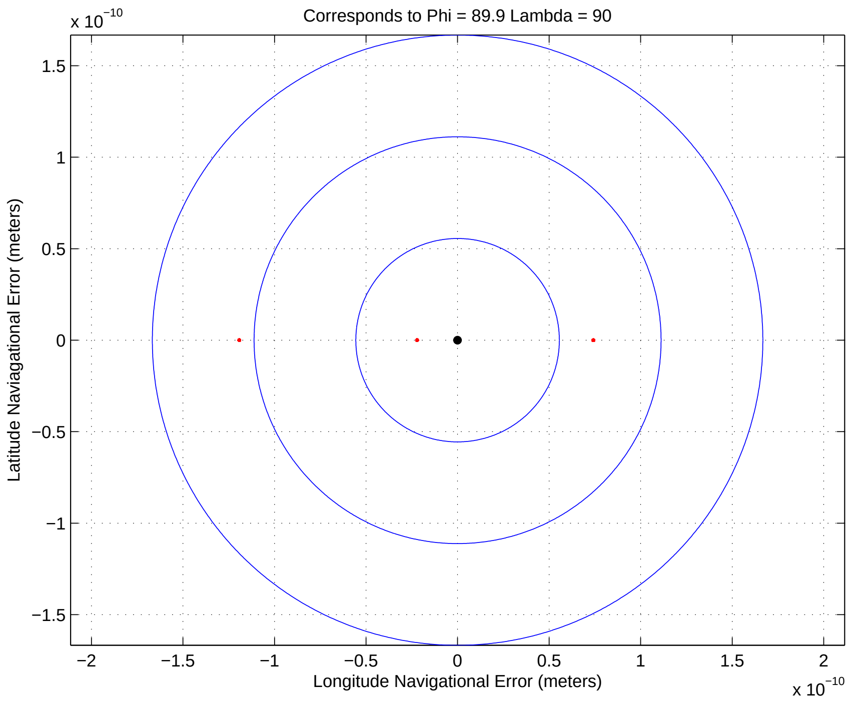

Figure 4.5: Case 1 (No Noise) - Latitude 89.9° Longitude 90°

Fig 4.6 shows an interesting occurrence at such extreme latitudes shown in the above Table 4.7. Sometimes there are between 1-100 outliers who's errors are so great they do not grow the sigma circles. These outliers occur naturally in normally distributed data, which this is, where the observed data point can be up to twice standard deviation with the possibility of the point being up too three times the standard deviation. (Kalos) With these outliers removed it resembles the previous case 2 simulations Fig 4.5. In practice the mobility solution would compare the current position to an estimated position to calculate the error, since an estimator works to smooth out anomalous spikes in data, this point would likely be ignored.

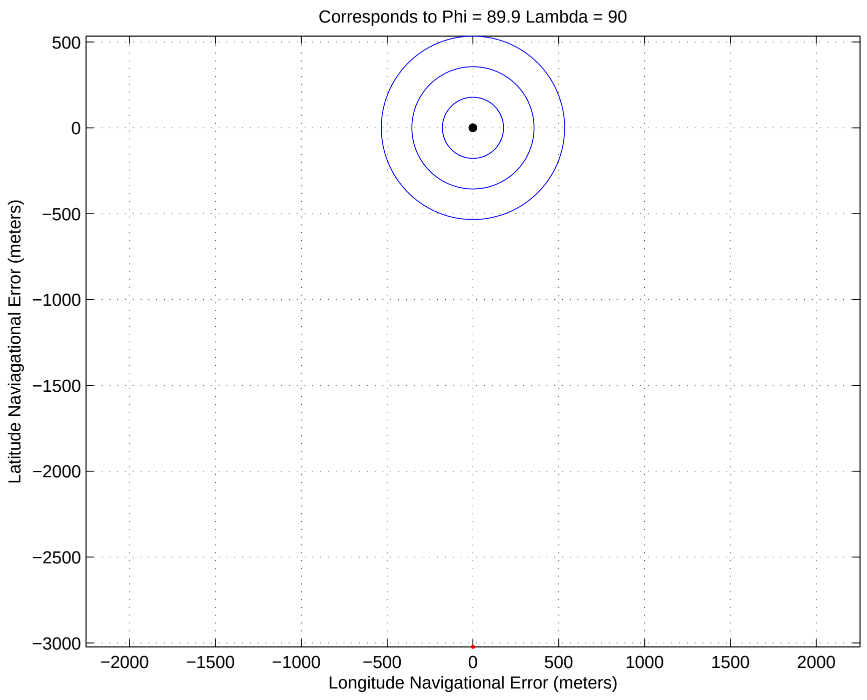

Figure 4.6: Case 2 (Accelerometer Measurement Noise) - Latitude 89.9° Longitude 90°

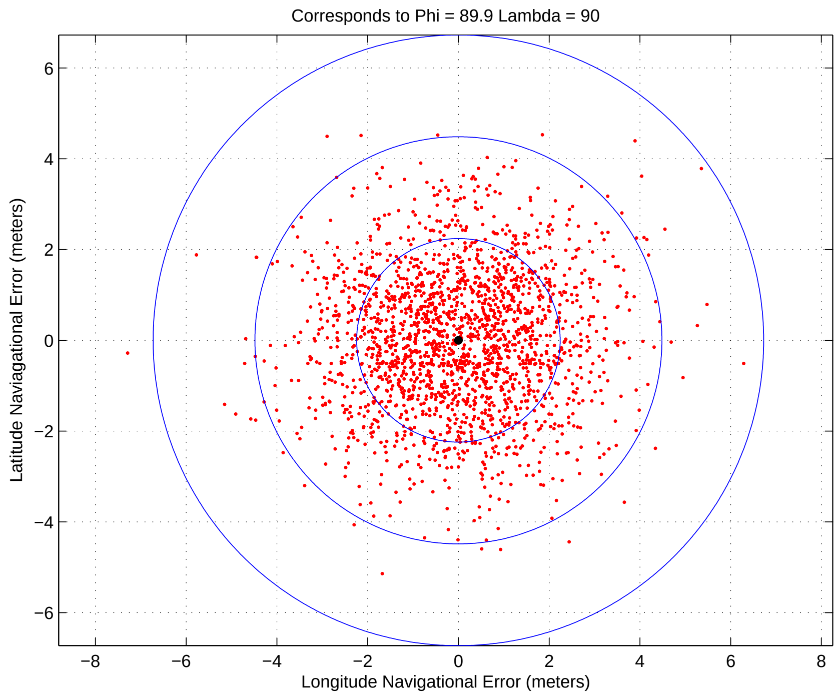

Figure 4.7: Case 2 (Outliers Removed) - Latitude 89.9° Longitude 90°

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

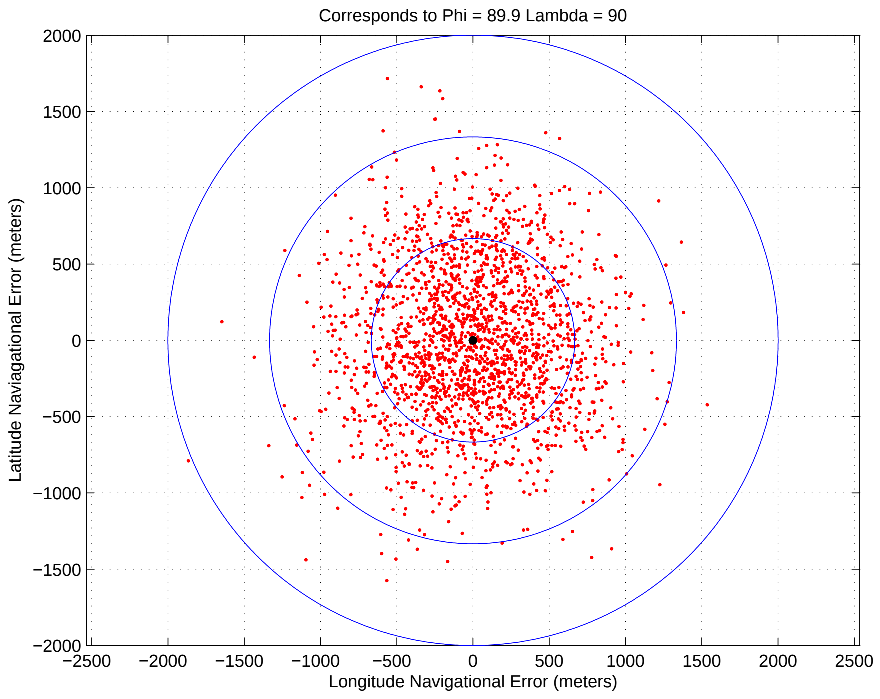

Figure 4.8: Case 3 (Worst Case Noise) - Latitude 0° Longitude 0°

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 case 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 Fig 4.9 the same visual can be seen as in the previous no error simulations, with errors on a scale of 10^{-11}, still showing small enough error to be considered numerical error.

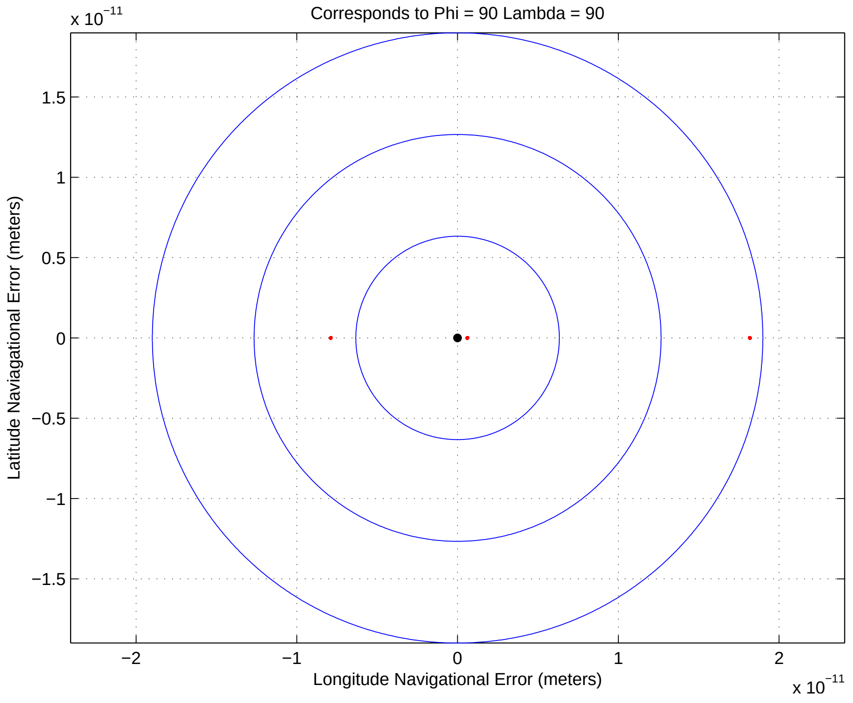

Figure 4.9: Case 1 (No Noise) - Latitude 90° Longitude 90°

Once noise is added to the systems it is possible to see an interesting effect. In Fig 4.11 there is no longitudinal error, this is because all longitudes converge at the poles as seen in Fig 4.10.

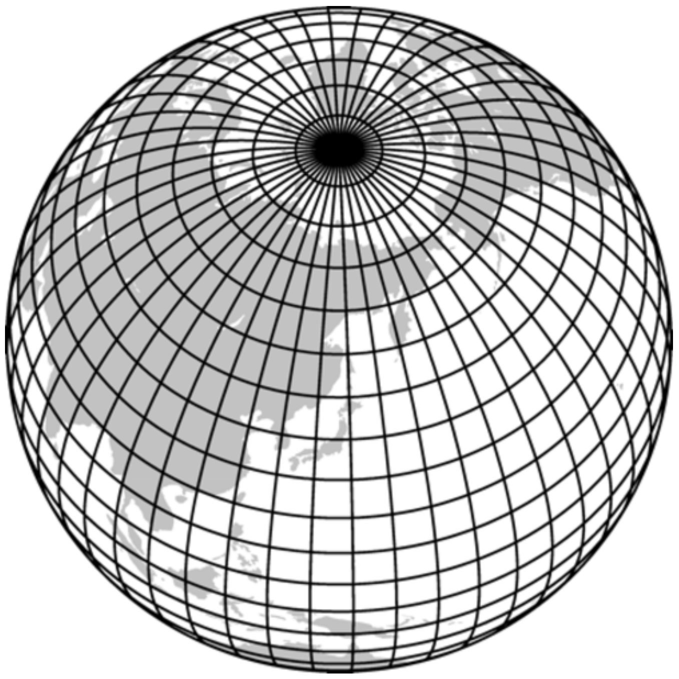

Figure 4.10: Latitude and Longitude Line Convergence

This means that in the polar region of a spherical body lines of latitude converge to a single point. Thus, the navigational error can be large, which means a small error in location (meters) can translate into a large navigational error. This is due to how close the lines of latitude exist to each other, so a small location error could 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 location error is very small extending only 4 meters.

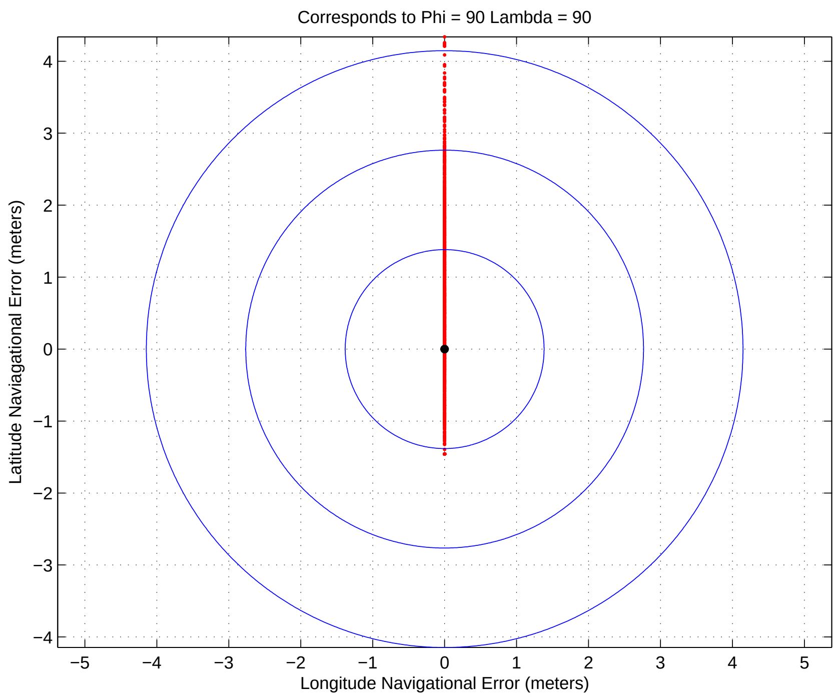

Figure 4.11: Case 2 (Accelerometer Measurement Noise) - Latitude 90° Longitude 90°

It can be seen in Fig 4.11 that the same form holds as seen above that there is no longitudinal error. But again using the worse case scenario noise the error jumps to 1500 meters with some samples falling over 1500 meters. (Recall that 2500 meters is at the upper limit of visibility on the lunar surface.)

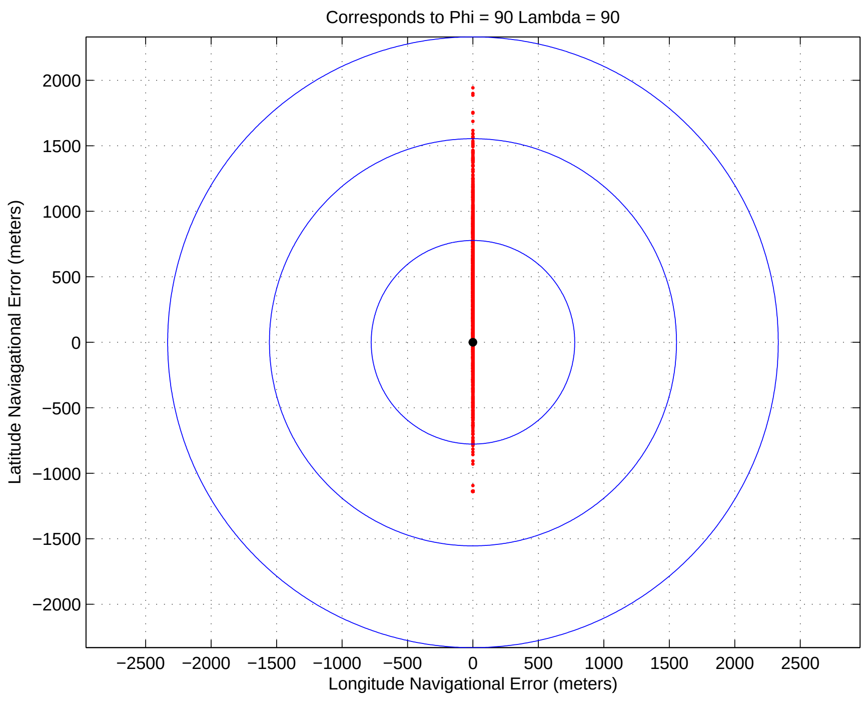

Figure 4.12: Worst Case Noise - Latitude 90° Longitude 90°

6 Conclusions

Monte Carlo analysis shows that over a large sampling (n = 2000+), there is a larger variance of possible valid locations based on the different levels of error, measurement and alignment. CelNav shows a gaussian distribution of results for all levels of noise, excluding no noise. The results shown above are very promising considering that analysis is a worst case un-calibrated simulation. 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 Moon.

Chapter V

Preliminary Experimental Validation of CelNav

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 that the ability to detect where it is point and thu, the celestial body the user is currently viewing, as well as give directions on how to orient one's body to view a specific 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, using proprietary software from the vendor, it is possible to extract raw data from the SkyScount. SkyScout outputs many different variable such as GPS coordinates, time (since 1-1-2000 available from the GPS signal), altitude, azimuth, rotation, right ascension, and declination. Using this information it is possible to construct all the necessary matrices (startracker (Delta) and Accelerometer (Gamma)) to run the CelNav algorithm.

The SkyScout is used to extract the raw data, startracker (Δ) and Accelerometer (Γ), necessary to run the CelNav algorithm. The initial results were run Greenbelt, Maryland. All the tests, 28 included, were performed in outside in a clearing, with the SkyScout mounted to a tripod for stability. Each test was performed in the same location at the same altitude, only the heading and the angle of the Skyscout were modified.

1 Developing CelNav Matrices

In this section, the Celeston SkyScout data is applied with the CelNav algorithm to confirm the algorithms legitimacy. Skyscout is only examined as a "stand alone" sensor platform supplying the measurement data necessary to run the CelNav algorithm. The SkyScout has the ability using proprietary software to output, latitude, longitude, elevation, Julian Time, altitude, azimuth, rotation, right ascension and declination. 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 these predefined matrices are examined against the "true" latitude and longitude extracted from the internal GPS unit.

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

Sidereal time is an astronomical time keeping 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 you can 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 (USNO). Sidereal time can then be converted into Greenwich Hour Angle (GHA), which measure the angle in degrees from any point on earth to the prime meridian.

In order to accurately transform from Earth-Center-Inertial (ECI), which does not rotate, to Earth-Center-Earth-Fixed (ECEF), which rotates with the earth, coordinates first the current number of days in Julian time must 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. (Winkler) This can then be converted into Sidereal Time.

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

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

This angle can then be converted into the transformation matrix of Earth-Center-Inertial to Earth-Center-Earth-Fixed,

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

This gives us the correct ECI to ECEF transformation matrix.

2 Constructing Startracker (Δ) and Accelerometer (Γ)

Extracting Δ and Γ is easily achieved since the SkyScout has the capacity to calculate the local latitude and longitude from internal sensors. 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 be compared to the “True” latitude and longitude from GPS data.

2.1 Extracting Accelerometer (Γ)

Gamma is relatively straight forward to derive as seen in Eq 5.4.

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

Where M_a is the accelerometer misalignment matrix and [ST] is the slope tilt matrix. M_a can be any valid misalignment matrix. [ST] can be constructed from the altitude and azimuth data such that:

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

These two easily defined matrices allow for the calculation of the Γ matrix.

2.2 Extracting Δ

Delta is relatively straight forward to derive as seen in Eq 5.6, 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:

$$[\mathrm{STH}] = \begin{bmatrix} \cos(\beta)\cos(\epsilon) - \sin(\beta)\sin(\alpha)\sin(\epsilon) & -\cos(\alpha)\sin(\epsilon) & -\sin(\beta)\cos(\alpha) \\ \cos(\beta)\sin(\epsilon) + \sin(\beta)\sin(\alpha)\cos(\epsilon) & \cos(\alpha)\cos(\epsilon) & \sin(\beta)\sin(\epsilon) - \cos(\beta)\sin(\alpha)\cos(\epsilon) \\ -\sin(\beta)\cos(\alpha) & \sin(\alpha) & \cos(\beta)\cos(\alpha) \end{bmatrix} \tag{5.11}$$

3 Results

Without loss of generality, two random test case studies, test case 14 and 28. These two were chosen at random and reflect overall what was seen using the Skyscout as the main sensor to for the CelNav algorithm. All raw data can be found in Appendix C.

It is assumed a max of 60 arcsec noise can applied to the alignment matrices of the accelerometer and star camera, this noise was artificially applied to the raw data.

Three different situations will be examined here including no noise, expected noise, and 60 arcsec noise. Note all noise will be evenly distributed about the x-y-z axis of each sensor, although in practice it is possible that one axis may be more heavily misaligned than another. 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 was not considered due to the use of real sensor data from the SkyScout.

3.1 Case 14

Case 14 is performed at Goddard Space Flight Center (Greenbelt, Maryland) (Quinn). It ss 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**
78.86°N	38.30°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 the alignment matrices it is possible to add error, in degrees, to each individual axis (x-y-z). The first test case, as seen in Table 5.2, has no added misalignment error.

	Trial 1 - 0 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	76.87°N	0	0
Longitude	38.81°W	0	
Heading	2.47°N	0	

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

As can be seen here there are no degree or position errors. This test shows that with good to low error measurements CelNav is able to return accurate data. Position error can be calculated as:

$$\begin{aligned}\text{Position Error} = \quad & \arccos(\cos(RA_{SS} * \tfrac{\Pi}{180}) * \cos(Dec_{SS} * \tfrac{\Pi}{180}) * \\ & \cos(RA_{CN} * \tfrac{\Pi}{180}) * \cos(Dec_{CN} * \tfrac{\Pi}{180}) + \\ & \sin(RA_{SS} * \tfrac{\Pi}{180}) * \cos(Dec_{SS} * \tfrac{\Pi}{180}) * \\ & \sin(RA_{CN} * \tfrac{\Pi}{180}) * \cos(Dec_{CN} * \tfrac{\Pi}{180})\end{aligned} \tag{5.12}$$

Where, RA$_{SS}$, is right ascension of the skyscout, RA$_{CN}$, is right ascension of the CelNav, Dec$_{SS}$, is declination of the SkyScout, and Dec$_{CN}$, is declination of the CelNav.

The next test case is with a expected misalignment noise, 20 arcsec of noise is chosen to represent expected levels of measurement noise. As can be seen in Table 5.3 the misalignment noise adds 0.002 degrees of error and 182.75m in navigational error. This shows that errors as small as 0.002 degrees results in large navigational errors.

The next trial incorporates what is considered as a high amount of misalignment noise.

	Trial 3 - 21 Arcsec Error		
	Location	Angle Error(Degree)	Navigational Error (meter)
Latitude	76.86°N	-0.002	182.75
Longitude	38.81°W	0.002	
Heading	2.48°N	0	

Table 5.3: Skyscout Case 14 Trial 2 - 21 Arcsec Error

As can be seen in Table 5.4 the misalignment noise adds -0.007 and 0.006 degrees of error and 548.24m in navigational error. Here it can be seen that the larger the misalignment noise the greater the latitude/longitude and navigational error that can occur.

	Trial 3 - 60 Arcsec Error		
	Location	Angle Error(Degree)	Navigational Error(meter)
Latitude	76.87°N	-0.007	548.24
Longitude	38.81°W	0.006	
Heading	2.47°N	0.001	

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

3.2 Case 28

The following tables show another test case, same location but facing in a different direction. The results are very similar to results previously discussed, the 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	78.86°N	0	0
Longitude	38.81°W	0	
Heading	90.69°E	0	

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

	Trial 2 - 21 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	78.86°N	0.003	182.51
Longitude	38.81°W	0.002	
Heading	90.69°E	0	

Table 5.7: Skyscout Case 28 Trial 2 - 21 Arcsec Error

In examining case 28 the results are quite comparable to those discussed in case 14.

	Trial 3 - 60 Arcsec Error		
	Location	Error(Degree)	Navigational Error (meter)
Latitude	76.85°N	0.008	
Longitude	38.81°W	0.006	547.54
Heading	90.69°E	0	

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

Some other test case data can be found in appendix X.X.

4 Conclusion

As it can be seen there is negligible error when comparing true latitude and longitude to the information derived from CelNav using the GPS information when no misalignment exists. As more misalignment error is added, and sensor noise is held constant, greater position error results. Even with as little as 0.001 degree error, this can translate into large navigational errors. The average error at the three different artificial navigational errors are for 0, 21 and 60 arcsec noise for case 14 and 28:

Table 5.9: Skyscout Average Error

There is, however, a disconnect from the way each system displays heading. Such that the Celestron SkyScout displays positive heading and CelNav displays both positive(180) and negative(-180) heading results. Such that, CelNav distinguishes between quadrants 1 and 4 as positive heading and quadrants 2 and 3 as negative heading, where as normal heading is measured clockwise, in degrees, from from north.

Chapter VI

CelNav Simulation(Experimental)

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.

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.

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 1. 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:

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

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 setup can be seen in Fig X.X. The CAST algorithm can be found in App. 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:

$$q = \begin{bmatrix} \sin(\frac{\theta}{2}) * \cos(\beta_x) \\ \sin(\frac{\theta}{2}) * \cos(\beta_y) \\ \sin(\frac{\theta}{2}) * \cos(\beta_z) \\ \cos(\frac{\theta}{2}) \end{bmatrix} = \begin{bmatrix} q1 \\ q2 \\ q3 \\ q0 \end{bmatrix} \tag{6.1}$$

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 $\cos(\beta_x), \cos(\beta_y), \text{and} \cos(\beta_z)$ 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:

$$\left[\ \Delta\ \right]_C^{Inertial} \longrightarrow \text{Star Tracker Measurement}$$

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 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.[OmniFlash]

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

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

Table 6.1: Possible light array setups.

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.

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○ 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.

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 mag-

	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 6.2: Sensor Statistics [Amy]

netometer as well as an additional GPS shield. [Amy] 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.[Amy] 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 equation 2.41. 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.[Amy] 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.[Amy] Please note this testing is to be a proof of concept for CelNav, and not the final experimental results.

As shown in Chapter 6 Underwood, it is possible using the IMU system her controllers would be compatible for integration with the CelNav platform to be used for navigation. Please refer to Chapter 6 and 7 in Underwood for a more indepth explanation.

Chapter VII

Simulation Model

The analytical rover model used win this research is that of Sun et al., which was originally the wheeled vehicle model was 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 a 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, a simple PID controller is used without loss of generality.

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.

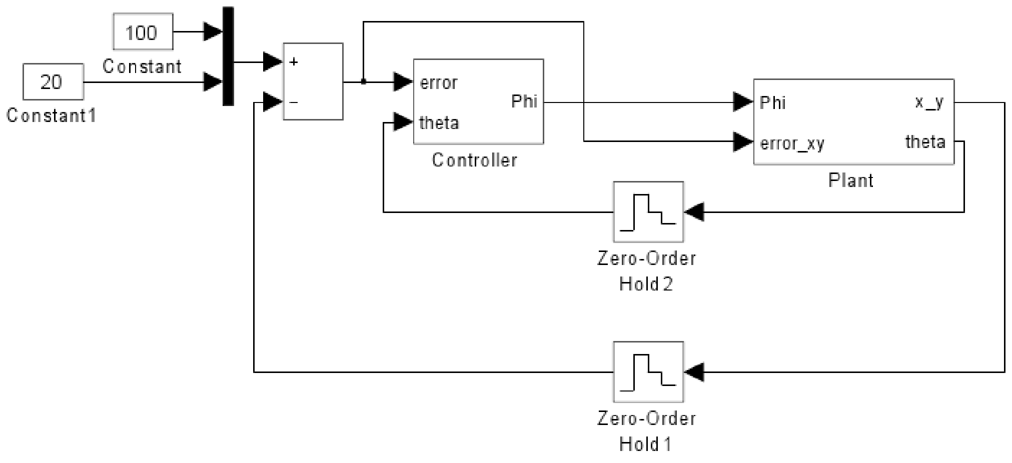

Figure 7.1: Simulation Model

First an overview of the vehicle dynamics as they appear in Eq (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 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 7.1: Variable from Rover Model in Sun et al.

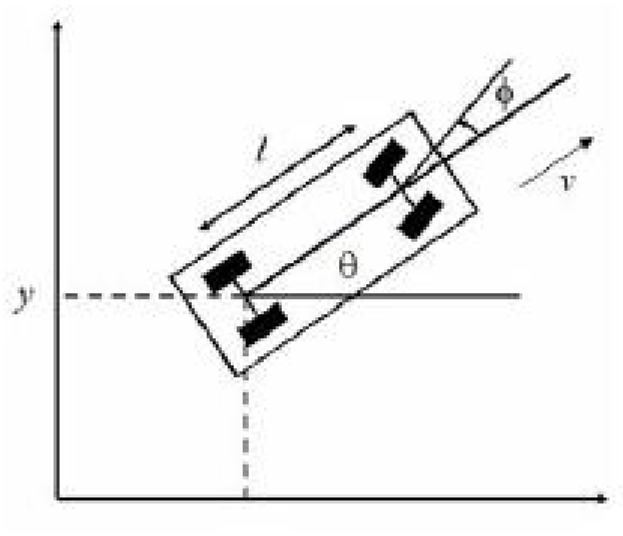

Figure 7.2: Simulation Model

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

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{a} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{l}\tan(\phi) \\ a \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \tag{7.1}$$

The above equation can be found in Sun et al. 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 Eq. (6.2).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{l}\tan(\phi) \end{bmatrix} \tag{7.2}$$

Velocity is now a fixed constant 0.1 $\frac{m}{s}$. The velocity term is found in Fig 6.2 under the block labeled "Vel In." This block controls the system velocity based on the x and y error. When this error decreased to within a predetermined threshold the velocity is turned off, 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 system. Even though the system is an inertial one the x and y are limited to 180° to -180° latitude and 90° to -90° longitude respectively to simulate longitudinal and latitudinal limitations.

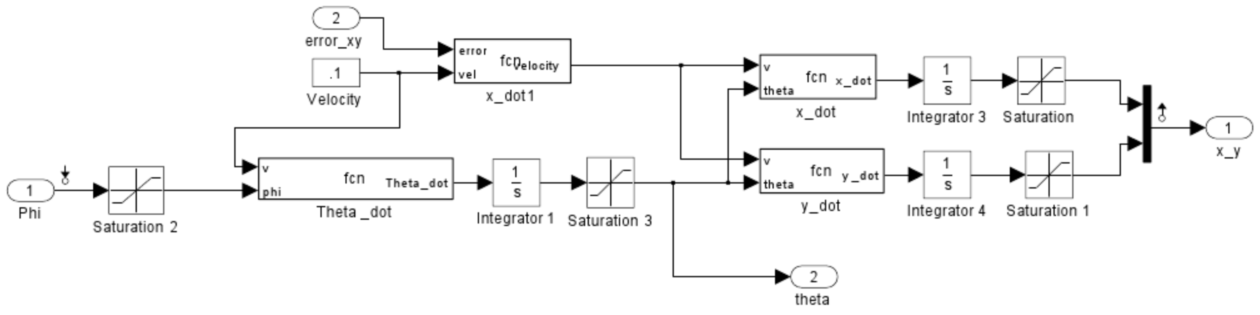

Figure 7.3: General Model for Rover

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 was 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.

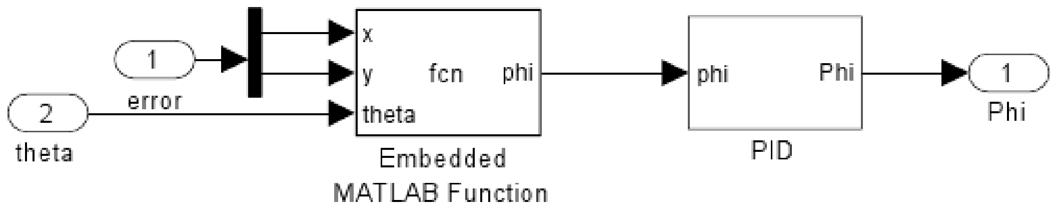

Figure 7.4: Controller Model

For this example only the position error, x (Latitude) and y (Longitude), are calculated as can be seen in Fig 6.3. Only heading (θ) is taken into account for feedback due to the added complexity to the PID controller, that is if the PID was to control x, y, θ. These gains can be found in Table 6.1. Again the main goal for this controller is stability, with a focus on the accuracy of the estimation techniques. As it can be seen there is no derivative term. This allows for a system with little to no overshoot.

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

Table 7.2: PID Gains

In order to control the system through heading it is necessary to calculate a heading correction factor (α). This is done by feeding back the heading as defined from the Plant Fig 6.1 back into the controller. This correction factor, α, can be found in Eqn 6.4:

$$\alpha = atan\frac{y}{x}; \tag{7.3}$$

By taking the previous heading (θ) and subtracting if from α the corrected heading term can is realized, as seen in Eqn 6.3:

$$\phi = \alpha - \theta; \tag{7.4}$$

Now that the correct heading term is known a new x and y position 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.

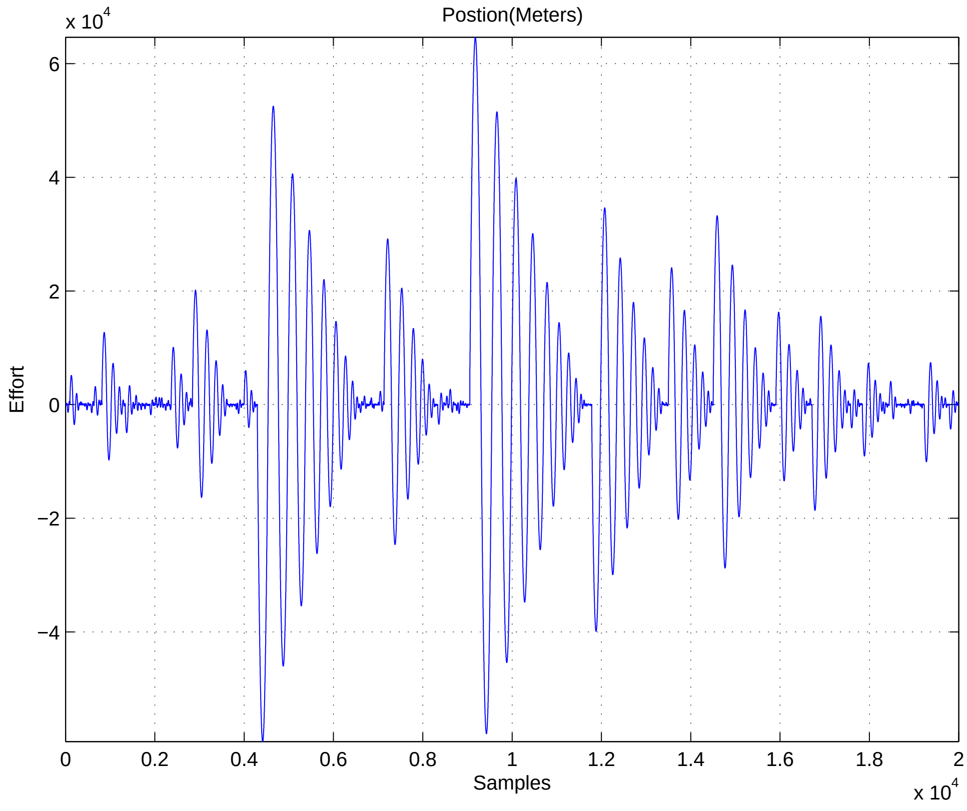

Figure 7.5: Control Effort

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. Attempts were made to reduce the magnitude of the control effort, but resulted in the inability to accurately control the system, sometime causing unstable conditions.

3 Results

For this simulation only accuracy and prevention of overshoot are deemed as the limiting factors. As can be seen in Fig 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.

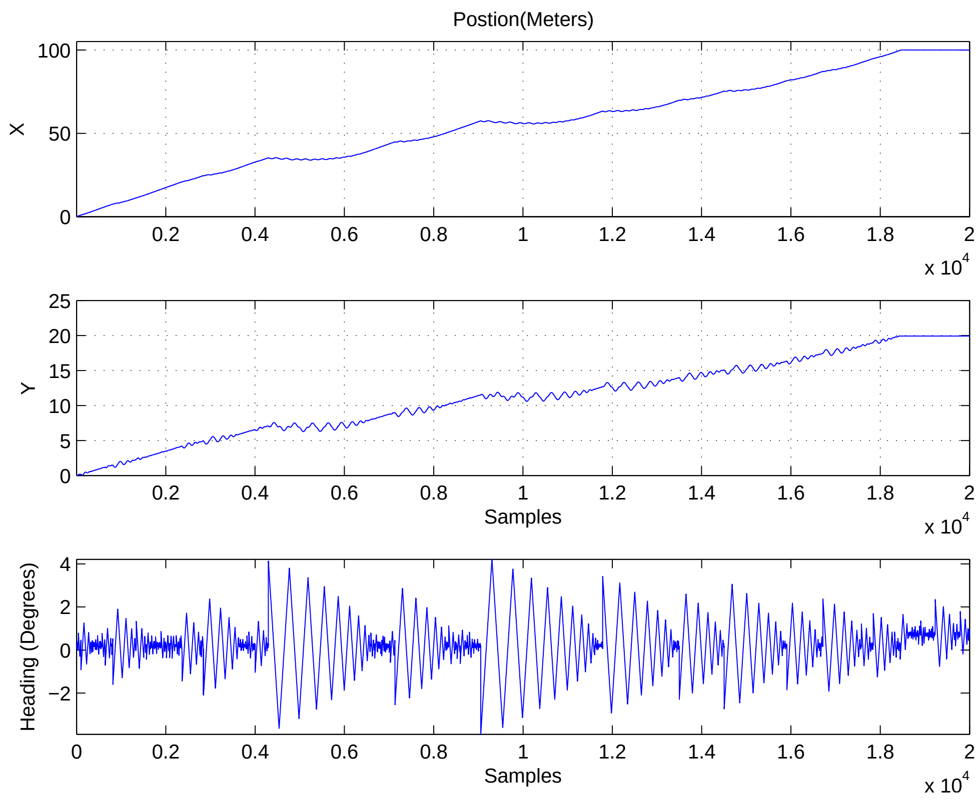

Figure 7.6: PID Results (X, Y, and Heading)

Heading can also be seen in Fig 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.

Below the x vs y can be seen in Fig 6.7. When Fig 6.7 compared to Fig 6.6 it is possible to see the same spikes in the x vs y trajectory as in the heading. This is do to the rover system trying to get to the desired location as quickly as possible, this happens to be almost a straight line.

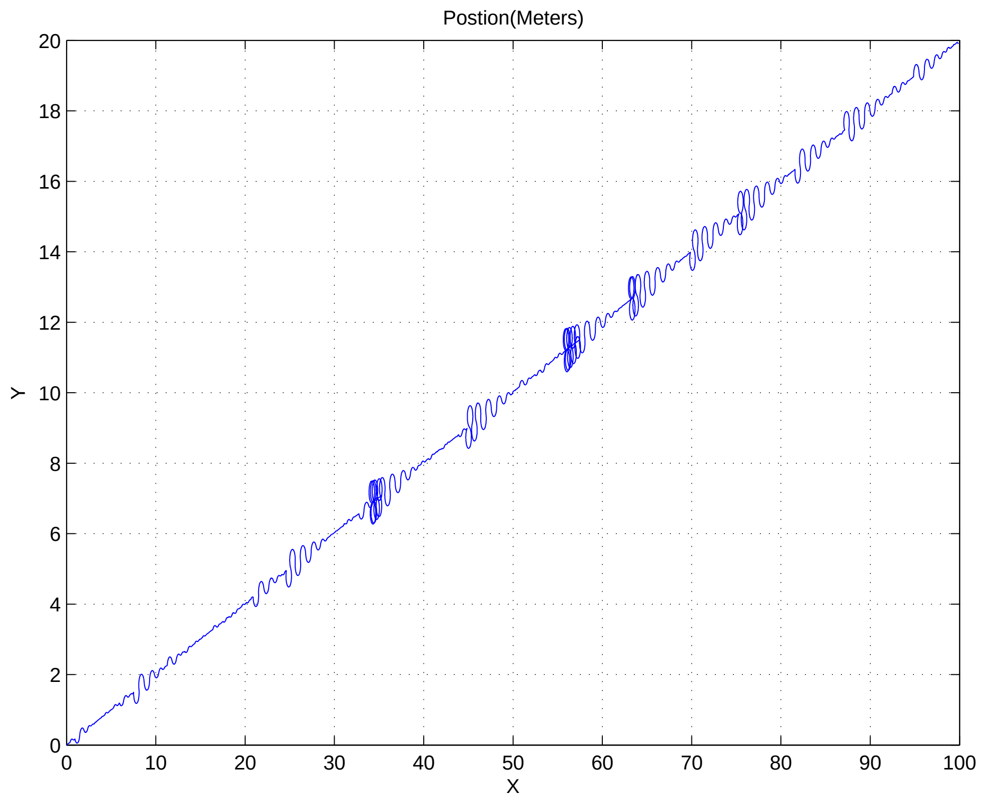

Figure 7.7: X and Y Trajectory

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

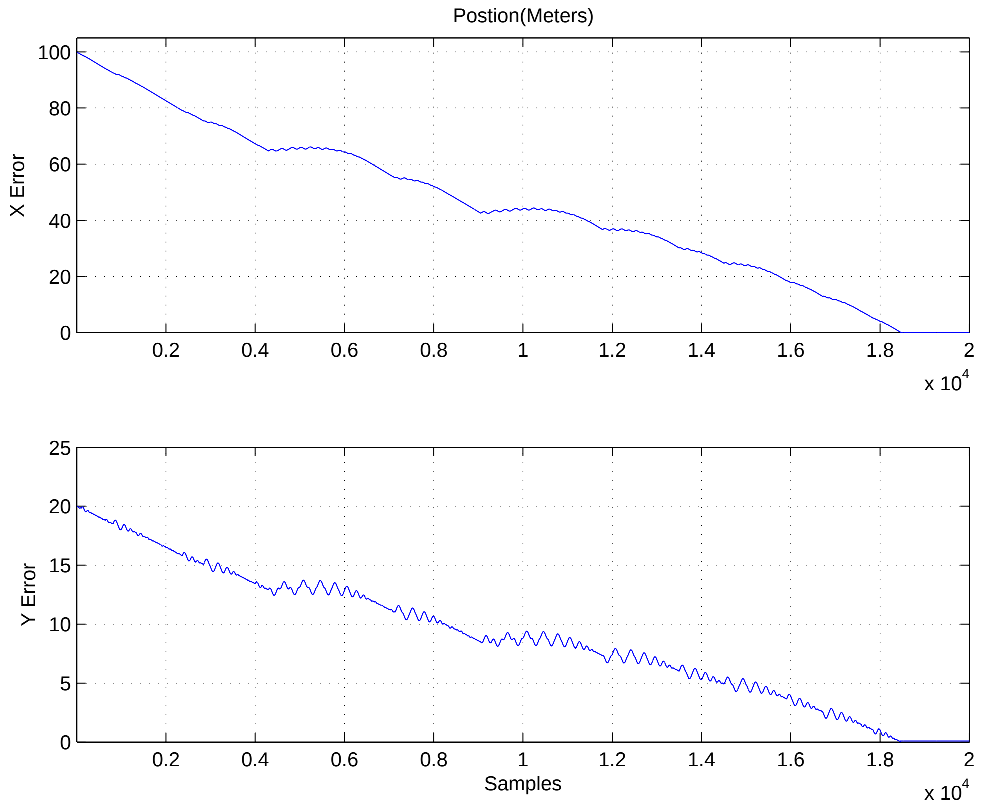

Figure 7.8: XY Error

It can be seen in Fig 6.4 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 will further be 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.

Chapter VIII

Theoretical/Analytical Results

This chapter will focus on both the theoretical and analytical results from different simulation conditions using three different estimation techniques, EKF, H-Infinity, and SMO. Each of these techniques will be analyzed individually with a total comparison to take place in a future chapter. Each will be analyzed with a focus on accuracy to true measurement, overall noise reduction and simulation efficiency. Additionally two different analysis will be looked at, first the viability of just the estimators with both rover models, then again with CelNav system integrated into both. This will allow for a comparison, to be discussed later, of how well the three different estimators handle possible erroneous and/or noisy CelNav reading. The first estimator that we will examine is the Extended Kalman Filter(EKF).

The nonlinear plant model is highly dependent on accurate measurements to be used in the control system, for this accurate estimations need to be made. 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 (θ heading), and a nonlinear plant model discussed previously. Only measurement noise will be investigated in the following chapters although alignment error has a significant effect on the CelNav algorithm.

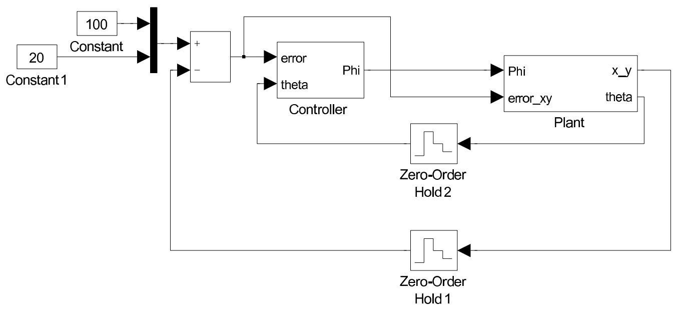

Figure 8.1: PID Model

Two different systems will be assessed; first the observers will be tested in the simulated environment without the addition of the CelNav algorithm to look at how well then work in high noise environments, then the second test will out how the observers using real sensor measurements and the CelNav algorithm to test the accuracy of the CelNav algorithm.

The rover based system is highly dependent on measurements with minimal noise residual. To accomplish this four different estimation techniques were examined; Luenberger Observer, Extended Kalman Filter(EKF), Sliding Mode Observer(SMO), and H-Infinity(H-∞). Each observer has different properties that are associated with the different observation techniques; these properties will be discussed at length in the following sections and the dynamics in future chapters.

1 Extended Kalman Filter

In this section the Extended Kalman Filter(EKF) for both computer simulations will be discussed. A comparison will be made between both the simulation results, looking at how CelNav works to provide accurate measurement results to supplement plant model state results. First simulation results will be examined followed by the CelNav experimental results with the EKF, then a comparison of both results will be assessed.

Optimal Estimation of Extended Kalman Filter works on the premise of returning the best estimated state with the least amount of error. As defined in Gelb(source) 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, as well as 3. initial condition information.

This method, though, is sensitive to incorrect or erroneous plant and measurement models. EKF can also be computationally overwhelming depending on the size of the system due to a matrix inversion that will be shown later. 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.

2 Governing Equations

Using the general nonlinear system and measurement equations found below:

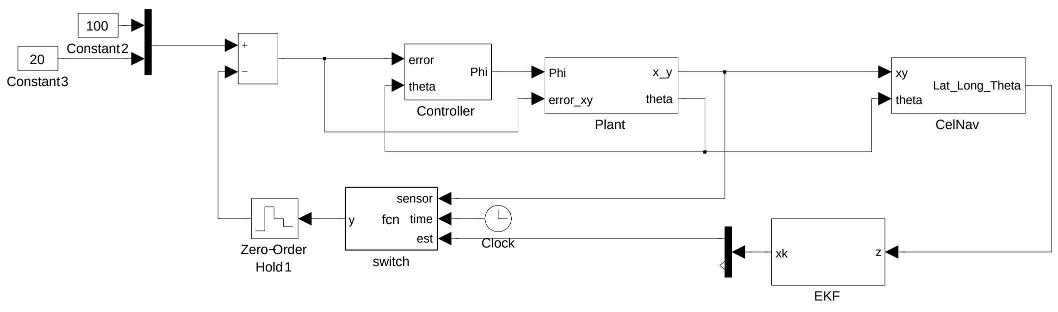

Figure 8.2:

$$\begin{aligned} x_k = & \quad F_{k-1}(x_{k-1}, u_{k-1}, w_{k-1}) \\ y_k = & \qquad H_k(x_k, \upsilon_k) \end{aligned} \tag{8.1}$$

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 follow using those contained in [Simon].

2.1 Kalman Filter Equations

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

$$F_{k-1} = \frac{\delta f_{k-1}}{\delta x}\Big|_{\hat{x}^+_{k-1}} \tag{8.2}$$

This must be updated at each time step to account for new x^+ values. Next it is necessary to update the estimate covariance matrix, this can be seen below in Eq 7.3.

$$P_{k-1} = F_{k-1}P^+_{k-1}F^T_{k-1} - L_{k-1}Q_{k-1}L^T_{k-1} \tag{8.3}$$

Along with updating the error covariance matrix it is necessary to update the state estimate as shown below:

$$\hat{x}_k^- = f_{x-1}(\hat{x}_{k-1}^+, u_{k-1}, 0) \tag{8.4}$$

The above calculations were done using previous time-step information; the following equations are done using both current measurements as well as the above updates. First it is necessary to update the Kalman gain as described in Eq 7.5.

$$K_k = P_k^- H_k^T (H_k P_K^- H_K^T + M_k R_k M_k^T)^{-1} \tag{8.5}$$

Here H_k is the matrix used to define what states we are wanting to estimate and M_k is assumed to be 1 due to unknown measurement noise models. Next, updating the state estimate is required. The only term that has not been shown before is $h_k(x_k^-, 0)$, This is a combination of the updated state estimate and H_k which are the states that we are wanting to estimate Eq 7.6.

$$\hat{x}_k^+ = \hat{x}_k^- + K_k[y_k - h_k(x_k^-, 0)] \tag{8.6}$$

The final step is to update the estimation-error covariance matrix.

$$P_k^+ = (I - K_k H_k) P_k^- \tag{8.7}$$

Choosing an initial P$_k$ of:

$$P_k = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix} \tag{8.8}$$

The above steps are repeated at each time step. Due this constant update a one sample time lag occurs meaning the estimates and the measurements are off by one sample.

3 Choosing Covariance Matrices

In the development on the EKF there are two separate covariance, the measurement noise covariance, R, and process noise covariance, Q. Each of these matrices are developed in two different ways. Measurement noise covariance, R, can be measured before implementation of the estimator with some number of unestimated simulations. Process noise covariance, Q, needs more work in order to obtain, since one may not know the actual system process noise.

3.1 Process Noise Covariance

$$Q = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.9}$$

$$Q = \begin{bmatrix} .7 & 0 & 0 & 0 & 0 & 0 \\ 0 & .7 & 0 & 0 & 0 & 0 \\ 0 & 0 & .7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.10}$$

(8.11)

3.2 Measurement Error Covariance

$$R = \begin{bmatrix} 5.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5.5 \end{bmatrix} \tag{8.12}$$

(8.13)

4 Simulation Results

This section will focus on the comparison of both normal noise conditions for the EKF simulation. Please note this section is to test the validity of the EKF observer on the defined rover system model being used. Two different simulations were run for each observer:

1. Normal Conditions
2. Worse Conditions

4.1 Normal Conditions

Results from the EKF as applied to the nonlinear simulated system with ”low” noise of $1 * 10^{-4}$ magnitude are shown below Fig 7.2 and Fig 7.3. A table of values can be found at the end of this section.

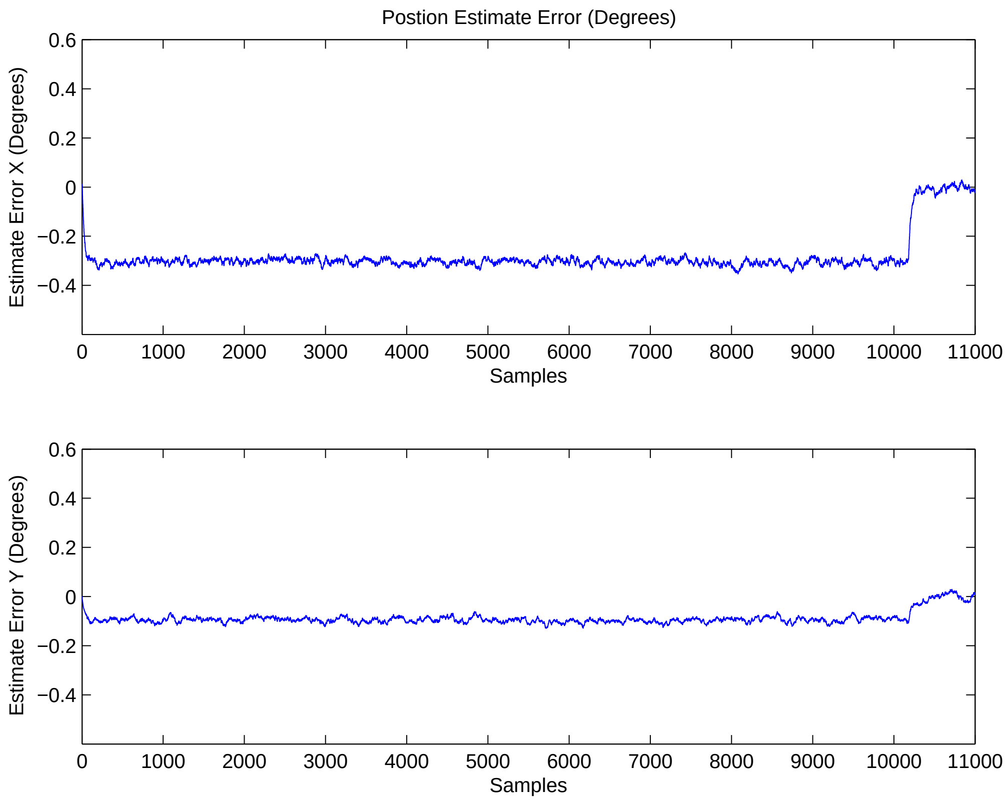

Figure 8.3:

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

Table 8.1: EKF Mean [μ] and Standard Deviation[σ] for "worse" noise

4.2 Worse Case Conditions

Results from the EKF as applied to the nonlinear simulated system with "low" noise of $1 * 10^{-2}$ magnitude are shown below Fig 7.4 and Fig 7.5. A table of values can be found at the end of this section.

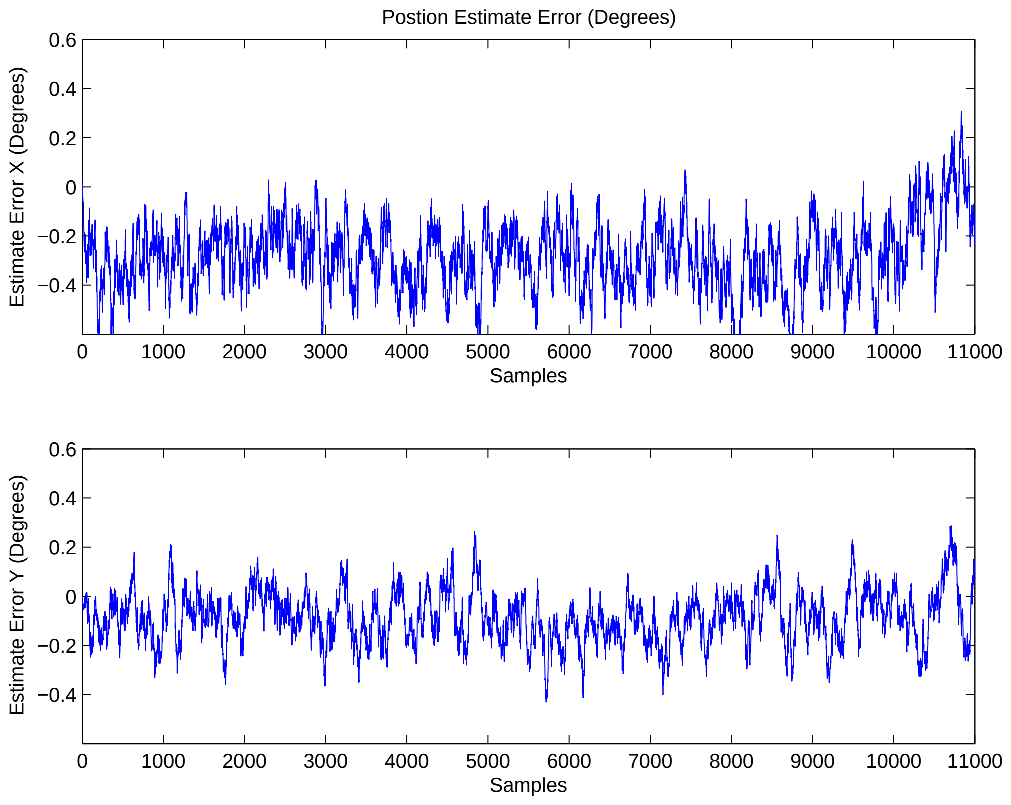

Figure 8.4:

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

Table 8.2: EKF Mean [μ] and Standard Deviation[σ] for "worst" noise

4.3 Simulated Error Computer Model Simulation w/o CelNav Algorithm

Below can be seen the analytical EKF simulation, without using CelNav. This is just a check of the ability of the estimator to help reduce measurement noise.

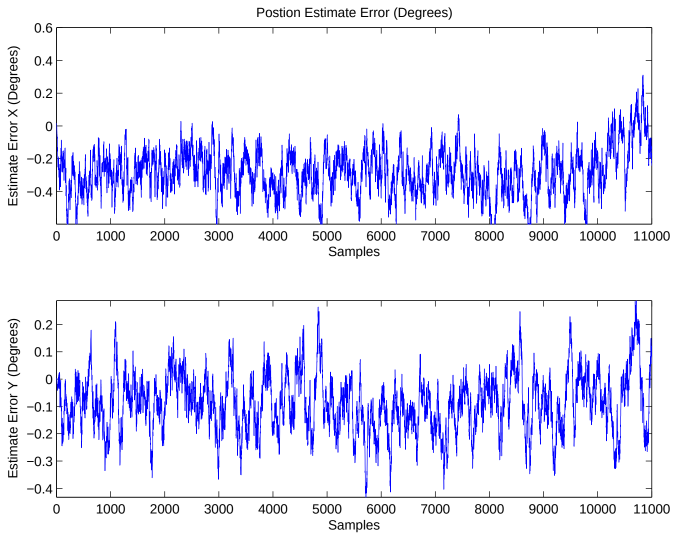

Figure 8.5: Computer Simulated Error w/o CelNav

5 H-Infinity(H_∞)

H-Infinity(H_∞) will be the next observer that will be examined. As stated earlier this method is very similar to the EKF method so results have the capacity to show similar results. The results will show that H_∞ observer as a most robust version of the EKF.

H_∞ was created as a way to add more robustness to the original Kalman filter as well as removing the need to known a priori statistics about system noise.(Simon) This type of observers is know as the worst case observer since certain aspect of the system may not be known, thus maximizing the cost function. It is possible to compare the Kalman observer to the H_∞, due to its nature the equations look a like an EKF, but for a few extra terms in the H_∞ observer. Thus it is possible to reduce the H_∞ observer to the Kalman observer based on the parameters choosen for the system. This method has been determined to be the most efficient way to create a robust Kalman Observer.(Simon) A more in depth analysis of the governing equations and methods will be given in a later chapter.

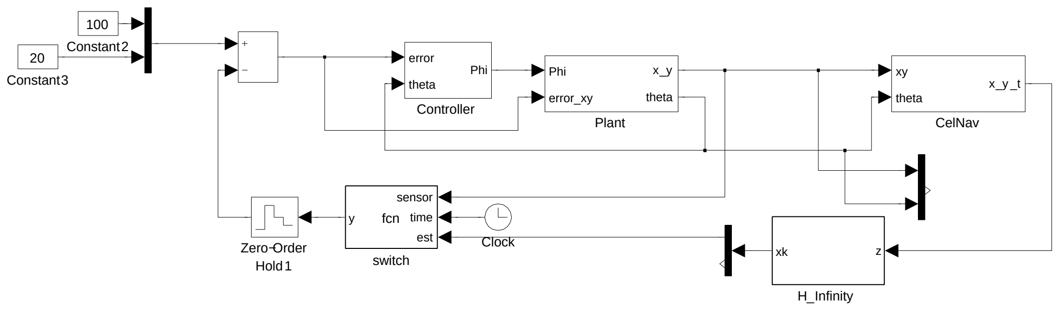

Figure 8.6:

6 Governing Equations

Following the linear discrete time model in the following explanation:

$$\begin{aligned} x_{k+1} &= F_k x_k + w_k \\ y_k &= H_k x_k + v_k \\ z_k &= L_k x_k \end{aligned} \tag{8.14}$$

[5]

Here w_k and v_k are both noise terms. z_k, as shown in the above equation, is the value that is to be estimated. L_k is user defined matrix necessary to estimate x_k. If L_k is assumed to be identity, full state estimation, then it would be equivalent using Kalman filter.

6.1 Cost Function

The H-∞ estimator is a cost minimization solution governed by the following cost function:

$$J_1 = \frac{\Sigma_{k=0}^{N-1} ||z_k - \hat{z}_k||_{S_k}^2}{||x_k - \hat{x}_k||_{P_0^{-1}}^2 + \Sigma_{k=0}^{N-1}(||w_k||_{Q_k^{-1}}^2 + ||v_k||_{R_k^{-1}}^2)} \tag{8.15}$$

Here z_k are the measured states, x_k is the estimated states, w_k and v_k are the noise terms associated with model and measurement uncertainties respectively. Also shown are S_k, P_0, Q_k, R_k which are user defined matrices that have the following properties: symmetric and positive definite. Our goal is too minimize $(z_k - \hat{z}_k)$ based upon our initial conditions for x_0 and our noise terms.

6.2 Estimation Functions

Here we will look at the equations that form the $H - \infty$ estimator. First we will look at Eq. 7.16

$$\bar{S}_k = L_k^T S_k L_k \tag{8.16}$$

This is the only non-relatable equation of the $H - \infty$ filter when compared to the Kalman filter. S_k can be defined as a symmetric positive definite matrix that is predefined by the user. It can also be shown that S_k will effect the K_k filter gains.

Here in Eq. 7.17 we see our second new term θ this is defined as our performance bound. The performance bound which seeks to minimize J such that $J < \frac{1}{\theta}$. It can also be seen that if $\theta = 0$ is chosen then the below equations become that of the Kalman Filter. These equations can be found below Eq. 7.17 - Eq. 7.19. They can be equated to the Kalman update equations, for the Extended Kalman Filter; here too they function in the same manner for the H_∞ filter.

$$K_k = P_k[I - \theta\bar{S}_k P_k + H_k^T R_k^{-1} H_k P_k]^{-1} H_k^T R_k^{-1} \tag{8.17}$$

$$\hat{x}_{k+1} = F_k\hat{x}_k + F_k K_k(y_k - H_k\hat{x}_k) \tag{8.18}$$

$$P_{k+1} = F_k P - K[I - \theta\bar{S}_k P_k + H_k^T R_k^{-1} H_k P_k]^{-1} F_k^T + Q_k \tag{8.19}$$

[Simon]

6.3 Minimization Solution

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

$$P_k^{-1} - \theta\bar{S}_k + H_k^T R_k^{-1} H_k > 0 \tag{8.20}$$

If at any timestep "k" this function does not prove true then the estimate for that time step is no a valid solution to the problem. For this above function to hold true and be positive definite requires that $\theta\bar{S}_k$. This can be achieved in three different ways:

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

7 Covariance Matrices

Below are shown the covariance choices for both the "low" and "high" noise H$_\infty$. When choosing covariance for the H_∞ filter it is usually necessary to reference your previous EKF. This is important because depending on how well the covariance matrices are developed for the EKF those quantities may be well defined and can be used for the H_∞.

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

$$L = \begin{bmatrix} .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & .1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.21}$$

and in the high noise setup Eqn 7.21 was used.

The same matrices should work for every level of noise with small alterations to account for less tracking of the noise and greater tracking of the signal. Comparing these choices to those discussed in the Chapter 5 the numbers for each case are quite similar differing by only small values. It is important to note that choosing R_k is normally related to the magnitude of the measurement noise; thus if your measurement noise is large the so should R_k; as with Q_k is similar with choosing Q_k for the EKF, increasing it to add robustness. This shows the compatibility of the H_∞ to the EKF, which allows for similar Q's and R's from to work.

8 Simulation

This section will focus on comparing both normal noise conditions for the EKF simulation. Please note this section is to test the validity of the H_∞ observer on the defined rover system model being used. Two different simulations were run for each observer:

1. Normal Conditions
2. Worse Conditions

8.1 Normal Conditions

Results from the H_$infty$ as applied to the nonlinear simulated system with "low" noise of $1 * 10^{-4}$ magnitude are shown below Fig 7.7 and Fig 7.8. A table of values can be found at the end of this section.

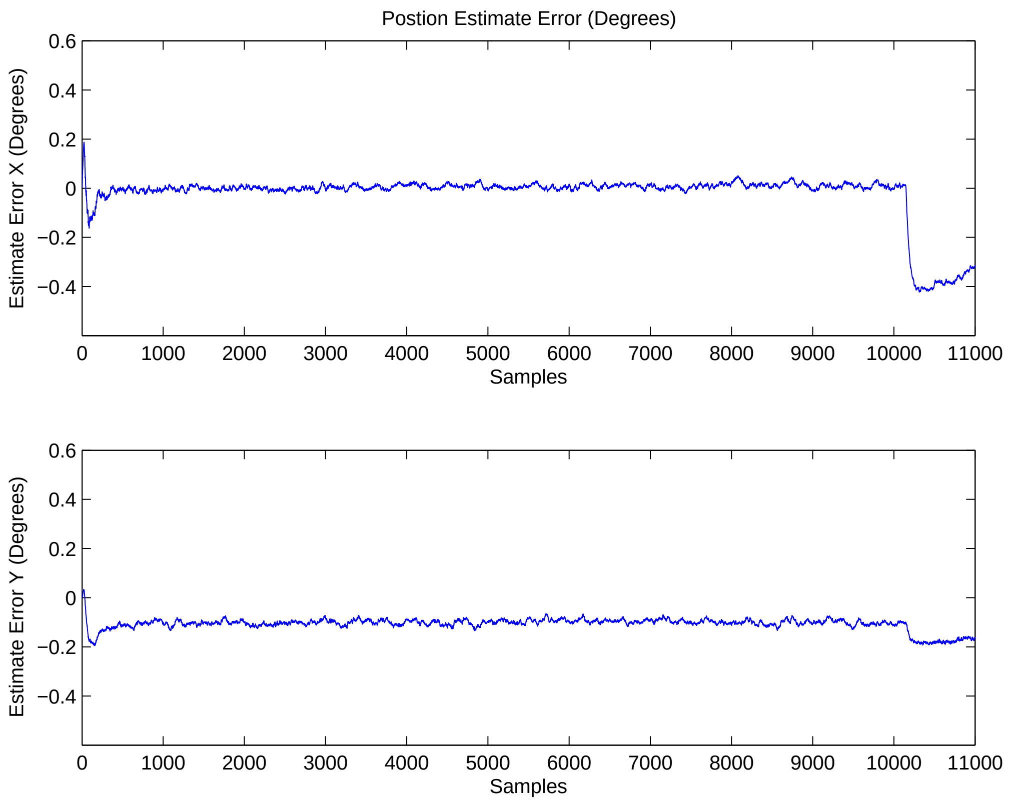

Figure 8.7:

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

Table 8.3: H-∞ Mean [μ] and Standard Deviation[σ] for "low" noise

8.2 Worse Case Conditions

Results from the H_∞ as applied to the nonlinear simulated system with "high" noise of $1 * 10^{-2}$ magnitude are shown below Fig 7.10 and Fig 7.11. A table of values can be found at the end of this section.

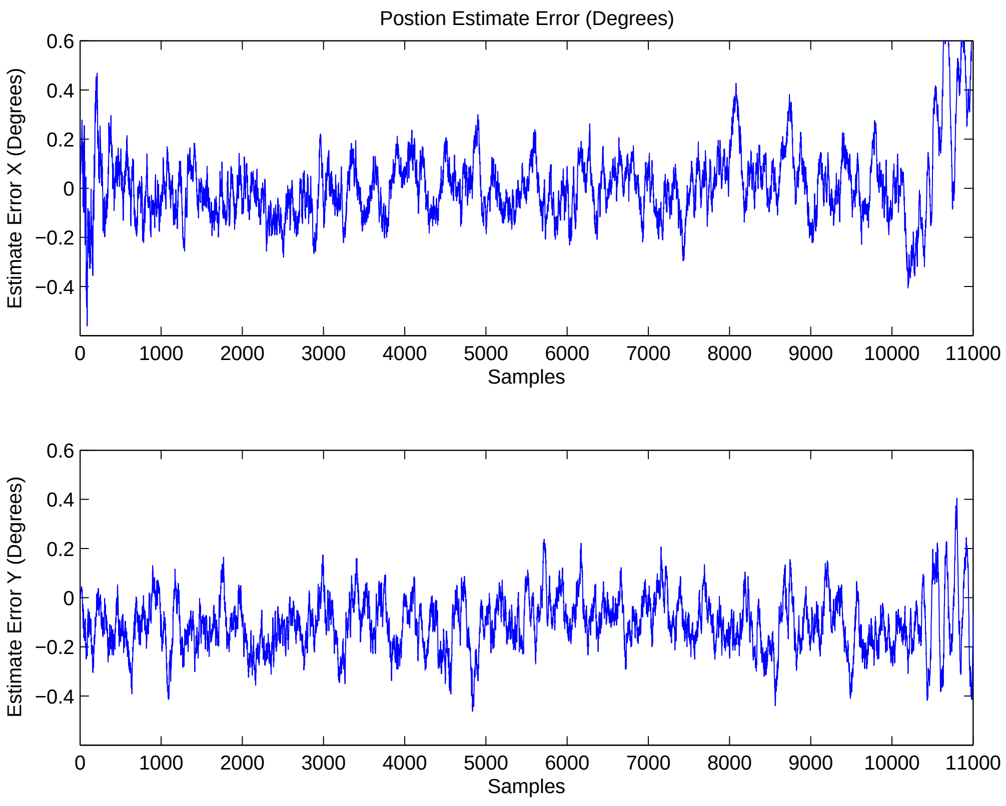

Figure 8.8:

Mean $[\mu]$		Standard Deviation$[\sigma]$	
Latitude	Longitude	Latitude	Longitude
0.0123	0.1045	0.1477	0.1101

Table 8.4: H-∞ Mean $[\mu]$ and Standard Deviation$[\sigma]$ for ”worst” noise

8.3 Simulated Error Computer Model Simulation w/o CelNav Algorithm

Below can be seen the analytical H-∞ simulation, without using CelNav. This is just a check of the ability of the estimator to help reduce measurement noise.

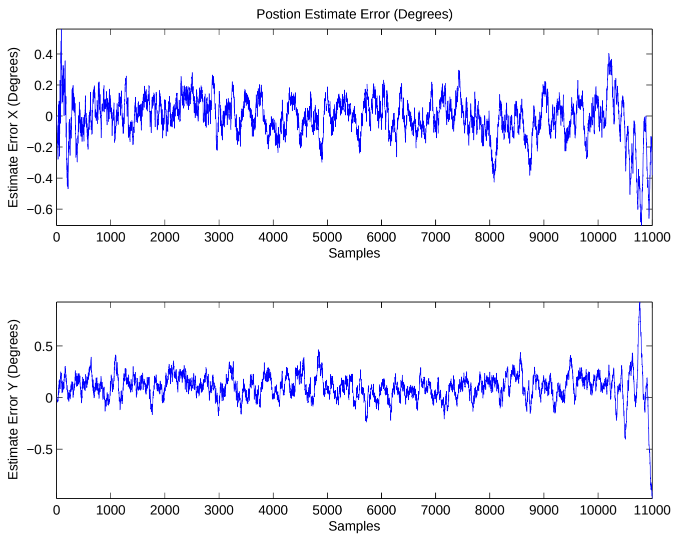

Figure 8.9: Computer Simulated Error

9 Sliding Mode Observer

In this chapter the Sliding Mode Observer(SMO) for both software simulation and experimental results will be discussed. Using CelNav the simulation and the experimental results will be compared and analyzed to see how the inclusion of CelNav to an observer can increase the accuracy of the estimated results.

Sliding Mode Observer(SMO) work on the assumption that it is easier to work with a 1st-order system then to work with an nst-order as defined by nst-order differential equations(Slotine/Li). SMO has the capability of providing excellent stability and performance when dealing with possible modeling errors.

SMO works in a two phase process know as the reaching phase and the sliding phase. (Reaching Phase). Next is the sliding phase. This part consists moving along an imaginary surface defined by a function s where the surface is defined as s=0. Once on this surface the function attempts to approach some defined position x$_d$. Switching to the path of this line is not instantaneous however,

10 Governing Equations

In this section the following linear system will be used to describe to SMO:

$$\begin{aligned} \dot{x}(t) = &\quad Ax(t) + Bu(t) \\ y(t) = &\quad\quad Cx(t) \end{aligned} \tag{8.22}$$

By defining the sliding surface, where s = 0, in the form of the below equation,

$$s = (\frac{d}{dt} + \lambda)\tilde{(}x) \tag{8.23}$$

$$
\begin{aligned}
sgn(s) = &\quad +1 \quad \text{if} s > 0 \\
sgn(s) = &\quad -1 \quad \text{if} s < 0
\end{aligned}
\tag{8.24}
$$

11 Gain Choice

Below is show the gain choice for both the SMO

Phi		
x	y	θ
0.01	0.01	2

Table 8.5: Phi Gains

	Gains		
	x	y	θ
K	1	1	.1
C	1	1	1

Table 8.6: Gains

Luenberger gains were found to have little to no effect on the SMO system, due to this the gains were left at 1.

$$
L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.25}
$$

12 Simulation

This section will focus on the comparing the both normal and worst case noise conditions for the SMO simulation. Please note this section is to test the validity of the SMO observer on the defined rover system model being used. Two different simulations were run for each observer:

1. Normal Conditions

2. Worst Conditions

12.1 Normal Conditions

Results from the SMO as applied to the nonlinear simulated system with ”low” noise of $1 * 10^{-4}$ magnitude are shown below Fig 7.10. A table of values can be found at the end of this section.

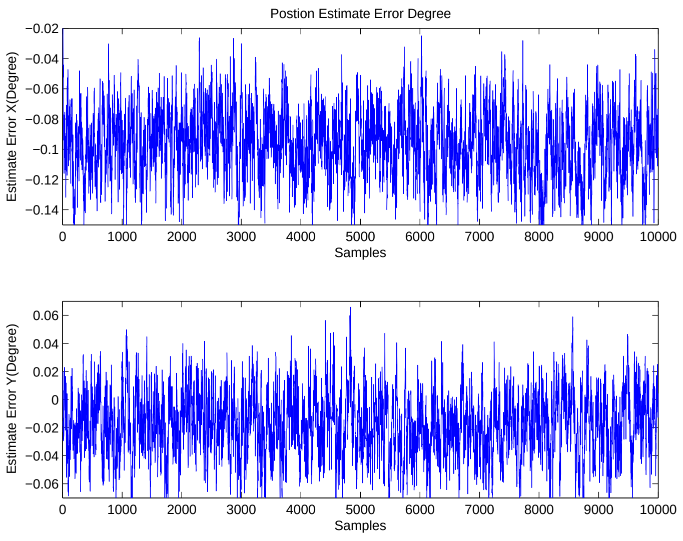

Figure 8.10:

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

Table 8.7: SMO Mean [μ] and Standard Deviation[σ] for ”Low” noise

12.2 Worse Case Conditions

Results from the SMO as applied to the nonlinear simulated system with "high" noise of $1 * 10^{-4}$ magnitude are shown below Fig 7.11. A table of values can be found at the end of this section.

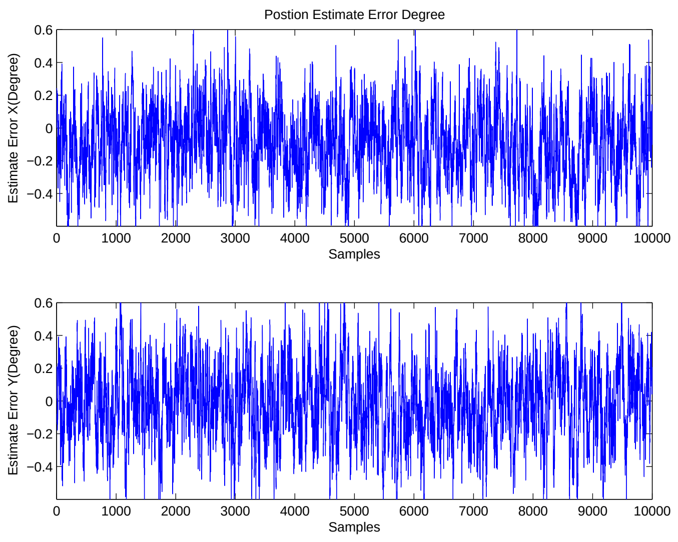

Figure 8.11:

Mean $[\mu]$		Standard Deviation$[\sigma]$	
Latitude	Longitude	Latitude	Longitude
0.0935	0.0175	0.0271	0.0229

Table 8.8: SMO Mean $[\mu]$ and Standard Deviation$[\sigma]$ for "Worst" noise

12.3 Simulated Error Computer Model Simulation w/o CelNav Algorithm

Below can be seen the analytical SMO simulation, without using CelNav. This is just a check of the ability of the estimator to help reduce measurement noise.

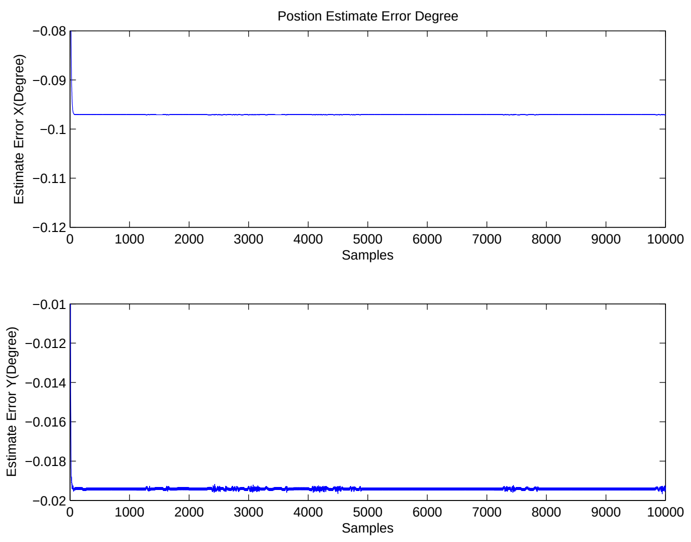

Figure 8.12: Computer Simulated Error

13 Comparison

This section will focus on comparing the three estimator's "normal" case conditions as well as the estimator's "worst" case conditions together, again only similar conditions will be compared. First the "normal" case conditions will be examined. The EFK as seen in Fig 7.3 (shown in degrees) it can be seen that there is a slight offset in the x error, this is due to trying to balance a noise free signal against system lag. in this case having an offset of approximately 0.2 degrees was deemed acceptable when when compared to have magnitude of the remaining noise on the signal which was approximately ± 0.05 degrees. The same can be seen in the y.

The EFK "worse" case conditions as seen in Fig 7.4 (shown in degrees), again there is a slight offset in the x error. Here the magnitude of the error was ± 0.1 degree. This is still a very small error in degrees.

Next the H-∞ estimator will be examined. Again as seen above in the EKF there is again an offset here again for the "normal" case, and again it can be seen that remaining signal deviation is ± 0.02 degrees. The same can be seen in the y.

Next the H-∞ estimator will be examined. Again as seen above in the EKF there is again an offset here again for the "worse" case, and again it can be seen that remaining signal deviation is ± 0.2 degrees. The same can be seen in the y.

Finally the SMO is examined. Again as seen above the "normal" and "worse" case, has a signal deviation is ± 0.02 degrees.

After examining each of these estimators individually as expected the EKF and H-∞ behaved similarly, but with the SMO having fastest simulation time. This is due to not having to perform and matrix inverse. All being said the H-∞ had the best performance of the three estimators. All of the estimators were able to converge and all were able to reach the desired location.

14 Conclusions

As was discussed above each of the different estimators behave with different characteristics, performance, speed, accuracy, ect, with the goal to find the optimal balance of each of these to produce the best results. As stated above the H-∞ had the best results while minimizing measurement error. It must almost be noted that both EKF and H-∞ did have an offset error since the system is running two timesteps behind, this however did not keep the system from reaching the desired location. The next section will discuss the results of the estimators in relation to how they respond to the experimental simulation.

Chapter IX

Conclusion

It is important to reiterate here that three (3) different overall topics were discussed:

1. Statistical Analysis

2. Experimental Setup

3. Analytical Experiments

These topics are important to building a proper foundation for current and future CelNav development.

The statistical analysis showed CelNav reacting extremely well at different levels of noise and different latitudes and longitudes. It was shown that even with exceedingly high levels of noise(example 10^{-6}) as shown in Table 4.6. The charts, shown in Chapter 4, are a good preliminary indication of the versatility of the CelNav algorithm. Even in the worse case scenarios the maximum error is 2000 meters, which again is at the upper end of the viewable area on the lunar surface.

As show in the Celestron experiment even small errors in degree, as little as .001 degree, can translate into large navigational errors on the scale of hundreds of meters.

The Celestron Skyscout is a great experimental sensor platform for preliminary real world experimental testing. 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 analytical experiments as shown were a great incite into how much computational power and memory are requires for both CelNav and the proposed estimators. This information is important for the future work of determining an effecting controller that will also be integrated into the simulation. These simulations have shown that the H-∞ has the fastest response time and the least residual noise. These results could be further improved upon with the addition of a tuned control system.

1 Future Work

Expansion on the three (3) topics discussed above should first focus on repeating the statistical analysis with CelNav and estimator, CelNav and controller and CelNav, estimator and controller. These three (3) additional statistical analysis would greatly further research into furthering CelNav as a future navigational system.

Future experimental setup should focus on designing a known course either geographical location (outdoors) or build an internal testing space with a know designed star pattern that can be uploaded into a star tracker for testing. This experimental setup help to further test the robustness of the CelNav algorithm through use in many different landscapes.

The goal of this research it to further the research on the development of a extra-terrestrial mobility solution. This would be combining the estimators described here with a controller and a physical rover to further integrate CelNav into a complete mobility solution. This we be actively comparing the CelNav solution, using a star tracker and accelerometers, along with other sensors, to an actual GPS solution. Future work should include comparing different estimators and controllers together to find the best combination, along with CelNav, for a low memory, low power autonomous mobility solution.

Future work should focus on not just comparing the best controller/estimator combination for accuracy, but the total computational effort, controller effort, and ability for further expansion. This will result in on different combination of estimators and controllers that could be used in different conditions, both physical and geological. These different combinations could be combined to for a completely autonomous mobility solution.

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 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)
% 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 no
        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(
        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), De
        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'*
            R_Gamma_raw = R_Gamma__Body_sAcc'*R_Delta_tot(:,:,j)*R_Omega__SD_I'*M
            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) -
                noise_Gamma_vec = mag_noise_Gamma*(randn(3,1) - 0.5); % evenly
                %                 noise_Gamma_quat = mag_noise_Gamma*(2*randn(3,1
                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))

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                % 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)) >5e-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);
    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)*
    ENav_y_m(2,e) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(lambda_mR(i)*
    ENav_z_m(2,e) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(phi_mR(i)*pi/

    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/1
    eNav_y_m(2,i) = R_moon*(1 + error_radius(i)/R_moon*rad_fac)*sin(lambda_mR(i)*pi/1
    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_
% 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_
% 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_
% 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_
% 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,bo

    % 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) Da
                    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_
    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*p
            error_vert(i)      = error_phi(i)/180*R_moon*pi;
            error_radius(i)    = sqrt(error_horiz(i)^2 + error_vert(i)^2);
            error_rad_angle(i) = atan2(error_horiz(i),error_vert(i))*180/pi;

        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 Ve
        %cd(Folder3);

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

   % Returns to Actual Working Directory
        cd('C:\Documents and Settings\controls\Desktop\CelNav\CelNav Versions\NASA Rework wi

   % 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 Versio
        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
                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

        % 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 I
                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
                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

        % Changes Working Directory
            Folder5 = sprintf('C:\\Documents and Settings\\controls\\Desktop\\CelNav\\CelNav
            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 Rewor
    end;
end;
```

Appendix B

CelNav Models

A.1 PID

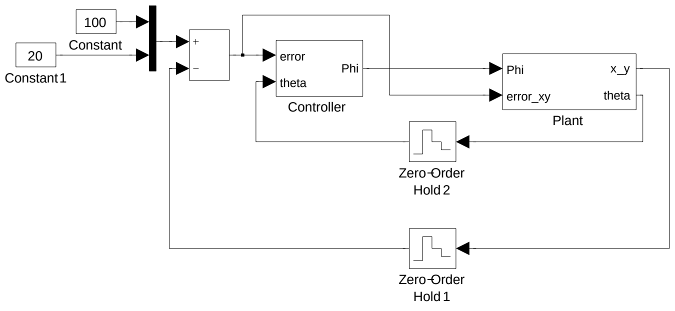

Figure B.1: PID Syster

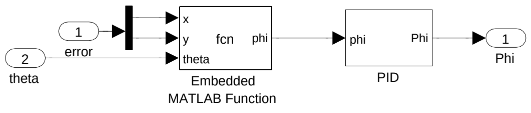

Figure B.2: PID Controller

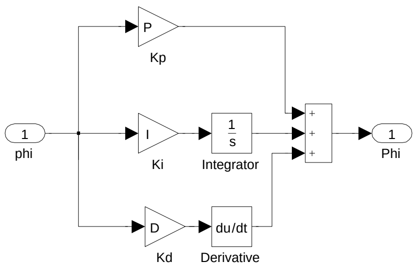

Figure B.3: PID Setup

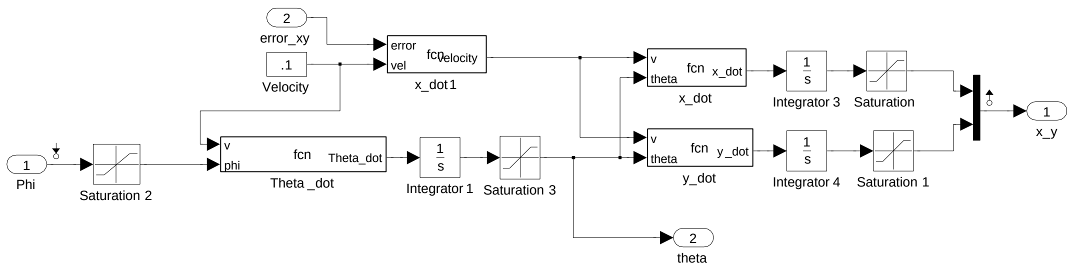

Figure B.4: Plant Setup (Rover Model)

1.1 Velocity Determination

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

1.2 Theta (Heading)

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

1.3 Velocity x-direction

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

1.4 Velocity y-direction

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

1.5 Heading Correction

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

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

A.2 EKF

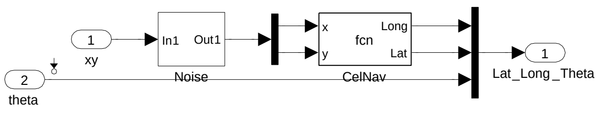

Figure B.5: CelNav Setup

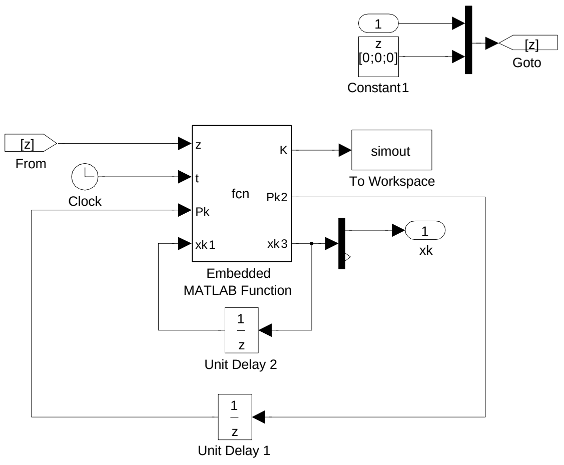

Figure B.6: EKF Setup

2.1 EKF Algorithm

```
    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
```

A.3 H-Infinity (h-∞)

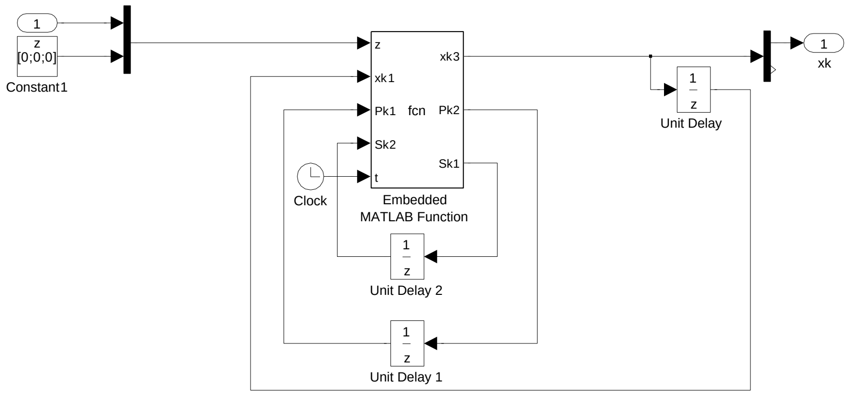

Figure B.7: H-∞ Setup

3.1 H-∞ Algorithm

```
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
```

A.4 Sliding Mode Observer (SMO)

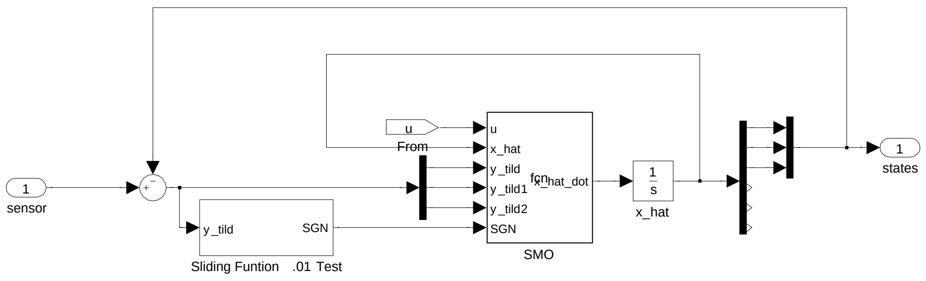

Figure B.8: SMO Setup

4.1 SMO Algorithm

```
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

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,'%

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,'%
Colors=['X Color: ' num2str(Color_xaxis,'%f'),'    Y Color: ' num2str(Color_yaxis,'%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

   % 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_EN



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%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 Rota
                                  R_Error_Body*R_Body*...            % Accelerometer Al
                                  R_M__C_Body'*R_M__C_Body_ae'*...        % Star Track
                                  Delta__C_Inertial'*...                  % NASA Model
                                  R_Omega__SD_I'*R_Omega__SD_I_E';        % Place Hold
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%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))

    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 D

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& mino
typedef int  (__stdcall *getGPSCmdTYPE)( double& latitude, double& longitude, double& elevat
typedef int  (__stdcall *getLastTargetCmdTYPE)( float& altitude, float& azimuth, float& rota
#ifdef USE_ORIENTATION
typedef int  (__stdcall *getOrientationCmdTYPE)(float& altitude, float& azimuth, float& rota
#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,outpu
% 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 E

Euler Angle Rotation Matrices (Direction Cosine Matrix)

A.1 Euler Angle Rotation Matrices

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)

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

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 Loc
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 LANDEI (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,J oaquim Alves Marine Sextant October 2006

[21] Brown, Hart "What is GPS?" Monday, May 13, 2013 http://www.mooreschools.com/page/19655