2013

Development of a Mars Exploration Rover for RASC-AL Exploration RoboOps Competition with an Extended Kalman Filter Based Navigation System

Benjamin J. Knabenshue
West Virginia University

Development of a Mars Exploration Rover for RASC-AL Exploration RoboOps Competition with an Extended Kalman Filter Based Navigation System

Benjamin J Knabenshue

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

Master of Science
In
Electrical Engineering

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2013

Keywords: Rover; Extended Kalman Filter; Inertial Navigation; Global Positioning System

Abstract

Development of a Mars Exploration Rover for RASC-AL Exploration RoboOps Competition with an Extended Kalman Filter Based Navigation System

Benjamin J Knabenshue

The primary objective of this research is to develop a navigation system for use on a rover designed for the NASA-NIA Revolutionary Aerospace Systems Concepts Academic Linkage (RASC-AL) Exploration RoboOps challenge. This competition takes place at NASA's Johnson Space Center Rock Yard, a test facility with many obstacles including steep hills, craters, rocks and loose sandy terrain. In addition to the challenges of the terrain, the rover's operators must control it from West Virginia University's campus. It was observed during the 2012 competition that a primary challenge was a lack of situational awareness; the operators had to navigate the large test area with only delayed video from two low resolution cameras. This research seeks to provide better awareness of the rover's position in the rock yard through the development of an accurate navigation assistance system.

This research first presents the rover that was designed, developed, and demonstrated at the 2012 competition that features six-wheel-drive, four-wheel steering, and a rocker-bogie suspension system similar to NASA rovers currently operating on Mars.

This research then presents a Navigation Assistance System (NAS) capable of providing nine-state navigation data, using small, lightweight, and low-cost sensors. In order to achieve an optimal estimate of the rover's attitude, velocity, and position this system employs an Extended Kalman Filter (EKF) that fuses data from an Inertial Measurement Unit (IMU) and a Global Positioning System (GPS). The EKF utilizes complimentary data from both systems to minimize the degradation caused by inherent system error that would be experienced by using either system alone.

The primary output of the NAS, position, is evaluated against a high-quality dual-antenna, Differential GPS (DGPS) receiver, as well as a low-cost single antenna GPS receiver. The NAS was observed to provide smoother position output, however actual error from the DGPS was only slightly reduced; in the nominal case an improvement of 0.2 meters was observed. Furthermore, testing demonstrated that the overall accuracy of the GPS alone, in comparison with the DGPS unit, was observed to be 1.7 meters. For the intended purposes of the system, it was found that using GPS alone would be sufficient for navigation applications.

The integration of this system with the rover brought unexpected challenges stemming from electrical interference created by on-board electronics, therefore this work presents the results of the navigation assistance system development taken off-board the rover. Furthermore, it presents research on autonomous navigation algorithms that could implement the state estimation data provided by the navigation assistance system that was developed.

Acknowledgements

It is with my utmost gratitude that I thank Dr. Powsiri Klinkhachorn and the 2011-2012 WVU Robotics team for their contributions to this body of work. Dr. Klink, as he is called by his students, has provided me with great challenges over my academic career along with the support and knowledge to successfully meet them. This work would not be possible without Dr. Klink and the rest of the team. Specifically, I want to thank Tim Godisart for his long hours, devotion, and major contributions to the design, fabrication, and wiring of the chassis. Thanks to Mark Pitts and Scott Zemerick for designing and building the core software that kept our rover running reliably and gave me a great foundation for my work. My thanks are extended to Dr. Gu and Tanmay Mandal for introducing us to the navigation system. Thanks to Jason Battin for providing some great designs and 3-D models for the rover and Abdullah Ibrahim for his Finite Element Analysis that helped us strengthen the chassis and ensure it would survive the rigors of the Johnson Space Center Rock Yard.

I would also like to express my sincere gratitude to Dr. Roy Nutter and Dr. Afzel Noore for contributing their time and expertise to this project by graciously sitting on my committee, and for providing valuable feedback contributing to the improvement of this work.

This project would not have been possible without the financial and logistical support of the NASA West Virginia Space Grant Consortium and specifically Dr. Majid Jaridi and Candy Cordwell. My sincere thanks go to the Benjamin M. Statler College of Engineering at West Virginia University and the Lane Department of Computer Science and Electrical Engineering for providing me with a great education and the student robotics team with necessary financial support to complete this project.

I thank NASA and the National Institute of Aerospace for sponsoring and organizing an inspiring and rewarding competition and for accepting our proposal for one of the eight available slots.

Thanks to my wife Traci and my parents Pat and Becky Knabenshue for their enduring support throughout my graduate academic career.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 Introduction

1.1. Background

The West Virginia University Mountaineers Robotics Team was honored to receive one of eight competition invitations to the 2012 Revolutionary Aerospace Systems Concepts Academic Linkage (RASC-AL) Exploration RoboOps challenge. The RoboOps challenge is a university competition in which teams are tasked to build a rover capable of collecting sample and traversing obstacle-rich terrain at NASA's Johnson Space Center (JSC) Rock Yard in Houston, TX, all while being controlled by operators stationed at each university's campus [1]. The competition provided the challenge of developing a rover that could handle obstacles, effectively collect samples, and stay operational for over an hour while being operated by a team at WVU's campus.

1.2. Problem Statement

One of the greatest challenges of the competition was controlling the rover with very little feedback, while experiencing communications delays averaging one second due to a wireless broadband communication channel that is mandated by the competition [1]. The operators stationed at the WVU campus in Morgantown, WV recognized a need for rover position telemetry as a result of the challenges of teleoperation over a slow communications channel. This presents the need for a navigation assistance system (NAS) to determine the Earth-based rover position, heading, attitude, and velocity for use in land navigation. The NAS should provide these data as telemetry to the rover operators and for future use in autonomous navigation. A further problem results from size and mass constraints set forth by the competition rules and financial constraints due to the team's budget; any system placed on the rover must be small, light, and low-cost.

1.3. Research Objectives

This thesis research seeks to design and develop a small, lightweight, low-cost NAS by fusing data from two small, lightweight, low-cost systems: an Inertial Measurement Unit (IMU) and a Global Position System (GPS). Furthermore, it will describe the development of a planetary rover capable of meeting all mission objectives set forth by the 2012 RASC-AL RoboOps *Planetary Rover Design Requirements* [1]. In order to accomplish this objective the following steps will be performed:

- Conduct a literature review to identify approaches to design a rover capable of handling harsh environments, approaches to state-of-the art autonomous vehicle navigation, and techniques for data fusion.
- Lead team to design, develop, and build a planetary rover prototype capable of competing in an academic Mars Rover competition.
- Develop and build a NAS that implements a nine-state Extended Kalman Filter (EKF) to fuse data from an IMU and GPS to achieve a high accuracy in position, attitude, and velocity measurements while keeping the NAS small, light, and low-cost.
- Verify results of the NAS through comparison to a high precision Commercial Off-The-Shelf (COTS) Differential GPS (DGPS) unit.

1.4. Scope

This research discusses the design and development of a NAS to be utilized in providing situational awareness to operators that are isolated from a rover under their control. The research explores the feasibility of using an EKF to fuse the data from small, lightweight, low-cost IMU and GPS systems to provide position estimates that are comparable to a much larger, more-expensive COTS DGPS solution.

In order to build a background and understanding of the system for the problem this research seeks to solve, this research discusses the hardware and software development necessary to build a rover capable of performing all tasks set forth by the 2012 RASC-AL RoboOps Competition [1]. It describes system highlights, design decisions, and verification of the rover systems.

1.5. Organization

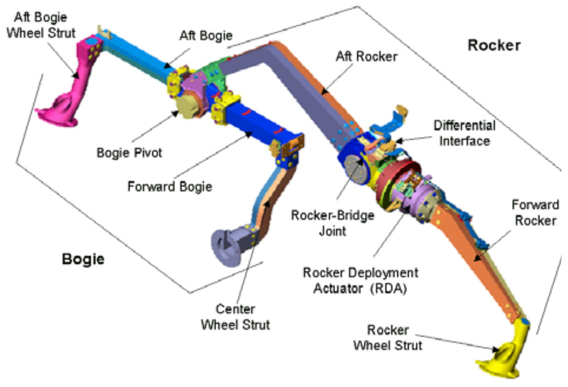
This thesis is organized into six chapters. Chapter 1 is an introduction including background of the problem, research objectives, and research scope. Chapter 2 details the literature review performed to identify known challenges in planetary rover design, navigation, current techniques used to navigate rovers, features of the IMU and GPS units selected for this research, and research on the EKF for data fusion. Chapter 3 describes the design and development of the rover platform used in the 2012 RASC-AL Exploration RoboOps challenge as well as the testing performed on the rover. Chapter 4 describes the design and implementation of the NAS, including a detailed look at the nine-state EKF used to determine the rover state. Chapter 5 describes the testing of the NAS and presents the results. Chapter 6 details the conclusions realized from this research and offers suggestions for future work. This thesis also includes appendices containing the source code for the research implementation.

Chapter 2
Literature Review

The development of planetary rovers is a challenging endeavor. Once beyond the atmosphere of Earth, considerations such as high radiation, extreme temperatures, lack of atmospheric pressure, and extremely long data transmission times must be accounted for. In designing a rover for the RASC-AL competition, considerations were made to account for the extraterrestrial phenomena that can be simulated on Earth such as expected obstacles, communications delays, and lack of atmospheric pressure. While the rover designed for the RASC-AL competition would not be suitable for extraterrestrial operations, the spirit in which it was designed reflects these considerations. Research on the NASA rovers was performed for inspiration for the design of the WVU rover and is reflected primarily in the chassis design; this research is presented in Section 2.1. While the WVU team did not have position feedback that could support autonomous navigation during the 2012 RASC-AL RoboOps competition, it is one of the primary topics of this thesis and a leading goal for the 2013 competition. The rover state development supports proposed future autonomous navigation based on the same techniques employed by the NASA rovers and terrestrial unmanned ground vehicles, with the exception of the use of Global Positioning System (GPS). This literature research is presented in Section 2.2. Section 2.3 describes the EKF and systems that employ the combination of a GPS and an IMU, combined with an EKF.

2.1. Planetary Rover Chassis Design

The NASA rovers *Spirit* and *Opportunity*, deployed for the Mars Exploration Rover (MER) mission, both feature a rocker-bogie suspension system as the foundation of their chassis [2]. This design features six individually driven wheels and four-wheel steering. This system does not use springs or pressurized elements such as hydraulics or pneumatics, making it more viable in most environments encountered during space exploration. The system is designed to evenly distribute the rover’s mass across all six wheels, minimizing sinking in soft and uneven ground. This is achieved by connecting two rocker-bogie assemblies, shown in Figure 2-1, to the rover body via a differential composed of two sets of epicyclic gearing contained in structures on opposite sides of the rover. The left side gearing is in a planetary gear configuration, and the right side is in a star gear configuration, shown in Figure 2-2. The two gear assemblies have opposite hands or rotations. This configuration creates a three-link differential motion with the rover body acting as ground. Thus, when the left side suspension goes up, the right side suspension goes down. This configuration equalizes the loads from the body to all six wheels, helping to minimize the ground pressure at any one wheel, therefore minimizing spinning and sinking of any one wheel [2].



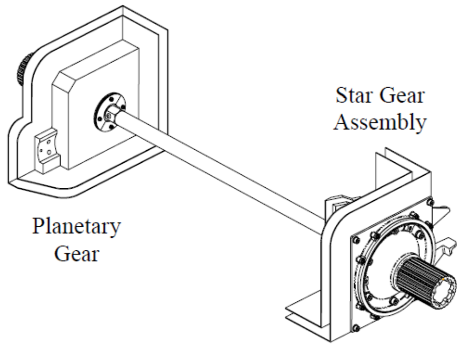


Figure 2-1: MER Rocker Bogie Suspension [2]

Figure 2-2: MER Rocker-Bogie Differential [2]

In order to maximize stability, the rover's center of mass is near the pivot point of the rocker bogie suspension. This allows the rover to tilt forty-five degrees in any direction without flipping over [2].

Each wheel in the rocker-bogie platform is independently actuated by its own DC motor. This provides six-wheel drive for maximum traction as well as redundancy in the drive system. This redundancy benefited the NASA rover *Spirit* in 2006. The right-front wheel stopped responding to commands, despite engineers' efforts to revive the wheel [3]. However *Spirit* was able to continue its mission, propelling with the five remaining wheels until April 2009 when it finally became stuck in Martian sands [4].

2.2. Autonomous Ground Vehicle Navigation

Autonomous navigation has been deployed on NASA's Mars Exploration Rovers, in various academic ground robotics projects, and even in automobiles in events such as the Defense Advanced Research Projects Agency (DARPA) Grand Challenge. These systems employ sophisticated algorithms to process a multitude of sensory systems and determine the correct actions to effect on the system in order to balance safety and navigation goals. The sensory systems provide information about the vehicle's position, trajectory, and operating environment. A sampling of deployed autonomous navigation schemes is presented in this section. Section 2.2.1 discusses the Mars Exploration Rovers' control scheme, which provided much of the inspiration for this project. Section 2.2.2 discusses Stanford's "Stanley" autonomous car, winner of the DARPA Grand Challenge. Stanley employed a combined Global Navigation Satellite System (GNSS)/Inertial Navigation System (INS) to provide trajectory information for navigation, providing inspiration for creating a rover that can autonomously navigate on Earth.

2.2.1. Mars Exploration Rover Navigation

Real-time teleoperation of the rover from mission control on Earth is not feasible. Large communications delays averaging 20 minutes are presented due to the average 225 million kilometer distance between the two planets [5]. At that distance, even the latest Mars rover, *Curiosity*, would require nearly 20 hours to send only 250 megabytes of data direct to Earth [6]. In order to best use the limited communications opportunities afforded by Earth's and Mars' orbits, the rover employs a sophisticated communications scheme, sending data to the Mars Reconnaissance Orbiter (MRO) when it passes overhead at a much higher rate than it is capable of communicating with Earth at. The MRO then sends data back to the Deep Space Network (DSN) on Earth. The DSN is an international network of antennas deployed in support of interplanetary spacecraft missions. Through the DSN, the rover is also capable of some communications directly with Earth when Mars and Earth are aligned. Even with these considerations, the rover requires a high degree of autonomy to operate [5].

The rovers are normally commanded once per *sol*, a Martian day. A sequence of control events are sent to specify the day's activities, the images and scientific data that are to be collected, and location goals. At the end of the sol's activities, images and data are transmitted to operators on Earth for analysis and to determine the next sol's activities [7, 8]. On the ground, dedicated teams examine the telemetry, including rover state data and images, and determine the next set of command sequences to send to the rover, illustrated in Figure 2-3.

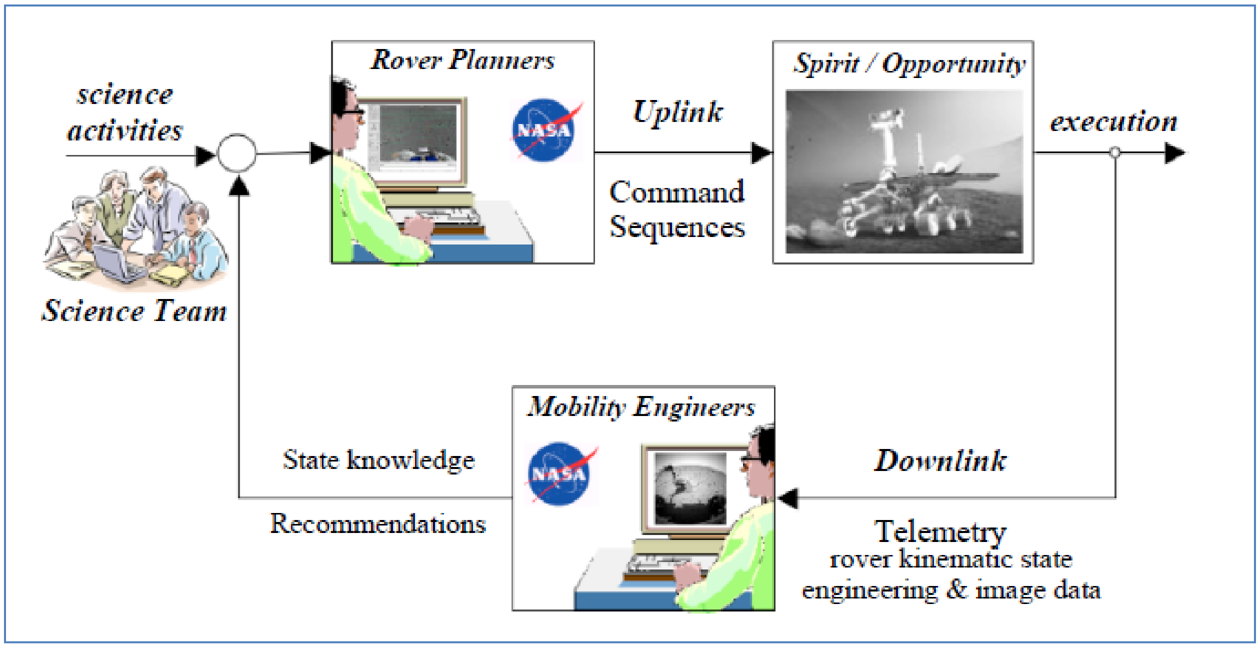


Figure 2-3: Overview of MER Mission Planning and Execution [5]

Once the ground team has provided the command sequences, the responsibility falls upon the rover's systems to execute them. The rover utilizes wheel odometry, an on-board IMU, and a suite of optical sensors to detect vehicle attitude and state [5]. The IMU and wheel odometry can be used in *dead-reckoning* navigation; however in many scenarios the wheels can slip, causing errors to grow unbounded on the dead-reckoning system. In order to correct the errors introduced by slippage due to steep slopes or loose terrain, visual odometry is used, however at a high processing expense [9].

2.2.2. Stanley: Autonomous Off-Road Vehicle

In 2004 and 2005, DARPA sponsored the Grand Challenge, an autonomous automobile competition. In 2004 no vehicles were able to complete the course; however in 2005 Stanford University's "Stanley" project won the competition by completing the 132 mile desert course in 6 hours and 53 minutes [10]. The vehicle travelled a course provided as a set of longitudes, latitudes, and corridors defined by maximum speed and width. An accurate estimation of the vehicle's state, including velocity, location, and attitude was necessary to stay on the dictated course. The vehicle used an IMU working in concert with a GPS compass and separate GPS location system to provide this data. In addition to the GPS/IMU system, the vehicle was also equipped with individual wheel speed encoders to provide better odometry in the event of GPS signal loss [11]. The Stanford team utilized an Unscented Kalman Filter (UKF) in order to fuse the data from each of the position sensors together.

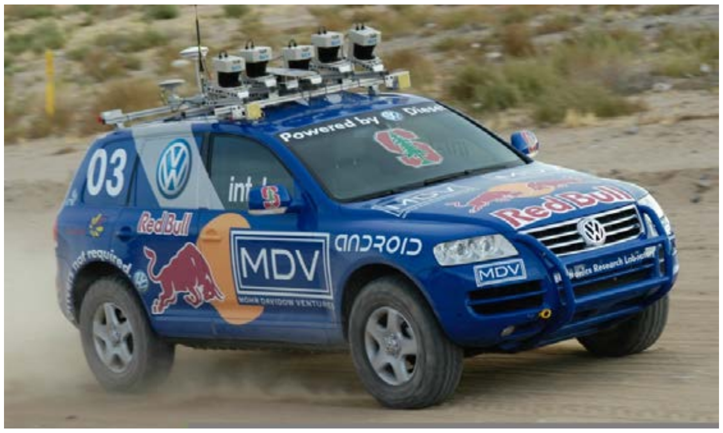


Figure 2-4: "Stanley" Autonomous Off-Road Vehicle [12]

In order to detect and avoid obstacles on the vehicle's course, "Stanley" is equipped with a suite of optical sensors. These included five scanning laser rangefinders, all pointed forward in the driving direction of the vehicle with varying angles of tilt for each sensor. This allows each sensor to measure a cross section of the terrain at varying distances in front of the vehicle with a maximum range of 25 meters [11]. Each laser rangefinder generates a vector of 181 measurements, each of them 0.5 degrees apart. In addition to the laser rangefinders, the vehicle is also equipped with two RADAR sensors capable of seeing obstacles up to 200 meters ahead of it with a coverage angle of approximately 20 degrees. These measurements are projected into a global coordinate system based on the state of the vehicle, resulting in a three dimensional point cloud for each laser rangefinder. The software then generates a two-dimensional surface grid and assigns each block a classification: *occupied*, *free*, or *unknown* based on the laser point cloud. *Occupied* indicates that two nearby points whose vertical distance exceeds a software-set threshold exist within the grid. *Free* indicates that at least one reading falls into the grid without exceeding the *occupied* test. If no readings fall into the grid, it is considered *unknown*. Once all grids on the two-dimensional surface are classified, the system creates a map containing a safe path, given one exists, for the vehicle to traverse. Accurate classification requires an accurate estimation of the vehicle's attitude, as even small errors in the vehicle's roll and pitch can create incorrect terrain maps [11].

2.3. GPS/IMU Sensor Fusion Using Extended Kalman Filter

GPS and INS based navigation systems have very complimentary error characteristics and lend themselves well to sensor fusion using a Kalman Filter [13]. Short-term position errors from the IMU are relatively small, but as their outputs are integrated to produce velocity and position feedback, these small errors grow without bound over longer periods of time. GPS position errors exhibit more variance in the short term, but are far more stable over time than the INS. The Kalman Filter takes advantage of the characteristics of each system, using the statistical information about the errors in both systems. This provides an integrated navigation system with performance superior to that of either subsystem (GPS or INS) [13].

The Kalman Filter has been identified as a fundamental tool for estimation and prediction problems. Section 2.3.1 introduces and describes the basic concepts of the Kalman Filter. Section 2.3.2 describes the adaptation of the Kalman Filter to nonlinear systems in the configuration commonly referred to as the EKF. Section 2.3.3 describes applications of the EKF and projects that were used to inspire this research.

2.3.1. Kalman Filter Introduction

The Kalman filter is a fundamental tool for solving a broad class of estimation problems [14]. It has been over fifty years since R.E. Kalman published his paper detailing the Kalman Filter [15] as a new approach to linear state estimation problems. Since then the Kalman Filter, in various forms, has been used in NASA's *Apollo* and space shuttle programs and in unmanned aerospace vehicles, to name a few examples [14].

The Kalman Filter is best considered as an algorithm acting upon a state-space model, including the state transition, observation, and process noise covariance matrices. At its fundamental level, the Kalman Filter predicts the value of the state variables periodically and then checks the system against new measurements. It updates the noise covariance for use in the next iteration to provide a better prediction.

Figure 2-5 shows an overview of the Kalman Filter, broken down into the major steps of the algorithm. This algorithm is well suited to sensor fusion, using outputs from different but related sensors to produce a better overall output. The output from one sensor can be used during the *prediction* step and then calibrated with another sensor's measurement provided in the *measurement* step.

Figure 2-6 shows the Kalman Filter algorithm in terms of the Kalman equations that define it. Steps two through five repeat; the current state prediction and covariance are utilized to determine the next state prediction and covariance. Table 2-1 shows the variables used in the Kalman filter equations.

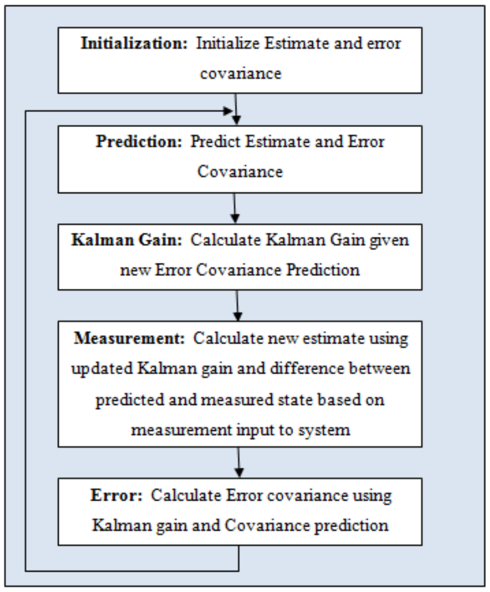

Initialization: Initialize Estimate and error covariance
Prediction: Predict Estimate and Error Covariance
Kalman Gain: Calculate Kalman Gain given new Error Covariance Prediction
Measurement: Calculate new estimate using updated Kalman gain and difference between predicted and measured state based on measurement input to system
Error: Calculate Error covariance using Kalman gain and Covariance prediction

Figure 2-5: Kalman Filter Overview

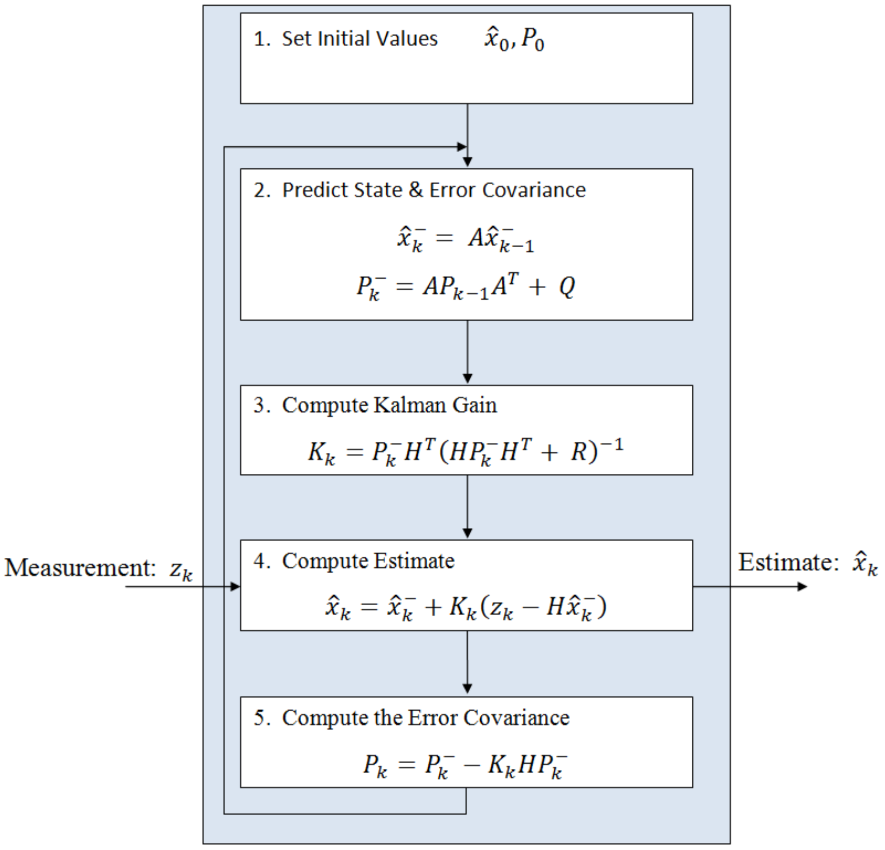

1. Set Initial Values $\hat{x}_0, P_0$
2. Predict State & Error Covariance
$\hat{x}_k^- = A\hat{x}_{k-1}^-$
$P_k^- = AP_{k-1}A^T + Q$
3. Compute Kalman Gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
4. Compute Estimate
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
5. Compute the Error Covariance
$P_k = P_k^- - K_k H P_k^-$
Measurement: z_k
Estimate: $\hat{x}_k$

Figure 2-6: Kalman Filter Overview Featuring Equations [16]

Table 2-1: Kalman Filter Equation Variables

Variable	Description
$\hat{x}_0$	Initial State
P_0	Initial Error Covariance
$\hat{x}_k^-$	Predicted State at time *k*
A	State Transition Matrix
$\hat{x}_{k-1}^-$	Predicted State at time *k-1*
P_k^-	Predicted Error Covariance at time *k*
P_{k-1}	Predicted Error Covariance at time *k-1*
Q	Estimated Error covariance of the Predicted State
K_k	Kalman gain
H	State Observation Matrix
R	Estimated Error covariance of the Measured State
$\hat{x}_k$	State Estimate
z_k	State Measurement
P_k	Error Covariance

2.3.2. Extended Kalman Filter

While initially conceived as a linear filtering approach, the necessity for use in real-world systems has produced Kalman Filter derivatives to handle nonlinear systems as well. One of the most notable, the EKF, was born of NASA's *Apollo* program through a fortuitous meeting of R.E Kalman and Stanly Schmidt in 1960 [14]. The EKF is widely used to handle nonlinear estimation problems [17]. There are higher-order approaches including second-order Kalman filtering, iterated Kalman filtering, sum-based Kalman filtering, and grid-based Kalman filtering. These methods involve more direct linearization of nonlinear systems, resulting in reduced linearization errors; however this comes at the cost of increased complexity and computational expense [17].

The simplest implementation of a Kalman Filter for nonlinear systems involves linearizing the nonlinear system around a nominal state trajectory and applying the standard Kalman Filter to this linearized system. While this works well for easily linearized systems, it does not work well for many real-world systems where estimating a nominal state trajectory is not straightforward [17]. In these cases the EKF is applicable. The EKF estimates the state of a nonlinear system by using the estimate of the system (output from the EKF) as the nominal state trajectory in a bootstrap fashion. The prediction $\hat{x}_k^-$ is defined as a function of the previous state estimate, and thus the prediction is steered by the estimate from the EKF. In addition, the state transition matrix **A**, previously static in each iteration of the Kalman Filter, now is the *Jacobian* of the system model functions, a dynamic matrix incorporating the partial derivatives of the state transfer function(s) evaluated at the current state estimate [16] [17]. This process is illustrated in Figure 2-7, the feedback from the estimate to the prediction and state transition matrix is denoted by a bold line. This approach allows highly nonlinear problems to be estimated with a Kalman Filter [16]. This approach applies to determining a vehicle's attitude in Euler space, using gyroscopes and accelerometers. This problem requires trigonometric functions to convert angular rates into body attitude, thus making it highly nonlinear. A further look at the use of an EKF to determine the pose of a vehicular body is presented in Section 2.3.3. Table 2-2 describes the variables used in the EKF equations.

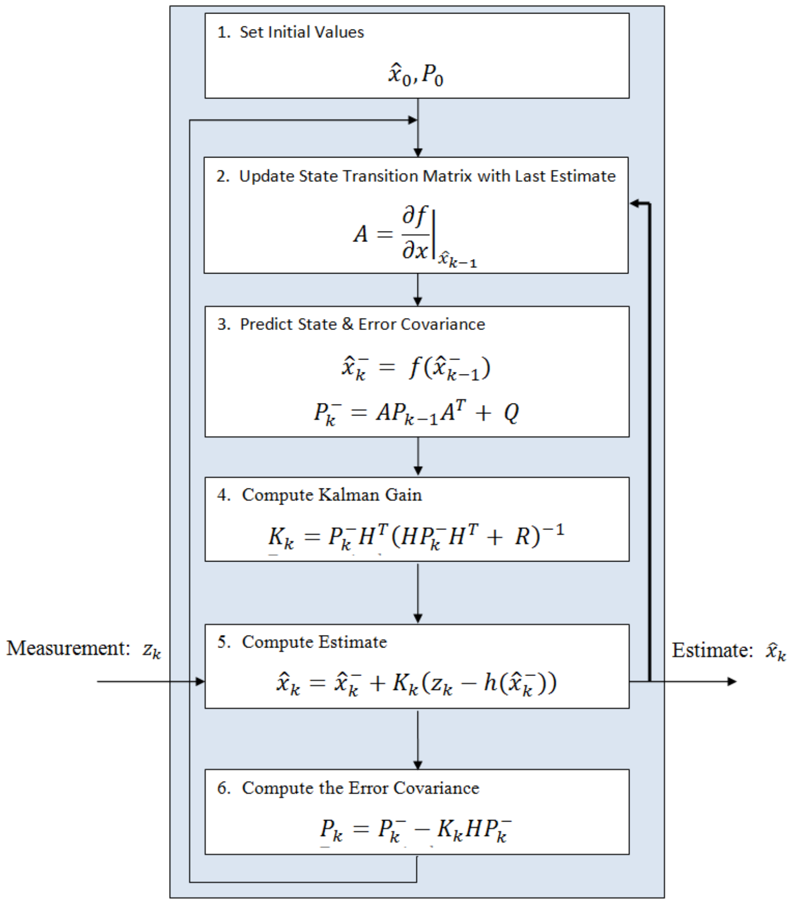

1. Set Initial Values
$\hat{x}_0, P_0$
2. Update State Transition Matrix with Last Estimate
$A = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1}}$
3. Predict State & Error Covariance
$\hat{x}_k^- = f(\hat{x}_{k-1}^-)$
$P_k^- = AP_{k-1}A^T + Q$
4. Compute Kalman Gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
Measurement: z_k
5. Compute Estimate
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h(\hat{x}_k^-))$
Estimate: $\hat{x}_k$
6. Compute the Error Covariance
$P_k = P_k^- - K_k H P_k^-$

Figure 2-7: Extended Kalman Filter Overview Diagram [16]

Table 2-2: Extended Kalman Filter Equation Variables

Variable	Description
$\hat{x}_0$	Initial State
P_0	Initial Error Covariance
A	State Transition Matrix, Jacobian of system model
$\hat{x}_k^-$	Predicted State at time *k*
$f(\hat{x}_{k-1}^-)$	System Model function of State at time *k-1*
$\hat{x}_{k-1}^-$	Predicted State at time *k-1*
P_k^-	Predicted Error Covariance at time *k*
P_{k-1}	Predicted Error Covariance at time *k-1*
Q	Estimated Error covariance of the Predicted State
K_k	Kalman gain
H	State Observation Matrix
R	Estimated Error covariance of the Measured State
$\hat{x}_k$	State Estimate
z_k	State Measurement
$f(\hat{x}_{k-1}^-)$	System Observation function of prediction
P_k	Error Covariance

2.3.3. Navigation Applications featuring Extended Kalman Filters

The fusion of gyroscope sensors and accelerometers to determine aircraft attitude is a classic application of an EKF. Gyroscopes provide a roll-rate (angle/time) and must be integrated to provide the angle of roll. Alone, gyroscopes give a great short-term indication of the roll with respect to the given axis, but due to integration, the error grows unbounded over time. Accelerometers measure acceleration with respect to the given axis, including gravitational acceleration and those caused by movement or change in attitude. Accelerometers typically have a bounded error, but are not as sensitive to changes in attitude, creating a good long term attitude estimate. The two sensors are complimentary to each other, and when combined in an EKF provide a better estimate of attitude than either sensor alone. Kim [16] details a simple attitude estimator in his text to demonstrate a practical example of the EKF. Kim's attitude estimator predicts the three axis attitude with gyroscopes and calibrates this estimation with data from accelerometers.

Attitude estimation utilizing an EKF is paramount to other state estimations such as velocity and position as seen in the work of Gu et al. [18] . This work proposed a low-cost aircraft navigation system that fuses GPS and IMU data to produce the three-axis position (*x*, *y*, *z*) velocity (V_x, V_y, V_z) and attitude estimates (roll, pitch, yaw) for an aircraft. This system predicts attitude by straight-forward integration of the roll-rates from the 3-axis gyro. Equation 1 shows the relationship of the predicted roll(ϕ), pitch(θ), and *yaw*(ψ) with the previous attitude state and gyroscope inputs(p, q, r) [18].

$$\begin{bmatrix} \phi^B_{k|k-1} \\ \theta^B_{k|k-1} \\ \psi^B_{k|k-1} \end{bmatrix} = \begin{bmatrix} \phi^B_{k-1|k-1} \\ \theta^B_{k-1|k-1} \\ \psi^B_{k-1|k-1} \end{bmatrix} + \begin{bmatrix} \tilde{p}^B_k + \tilde{q}^B_k \sin\phi^B_{k-1|k-1} \tan\theta^B_{k-1|k-1} + \tilde{r}^B_k \cos\phi^B_{k-1|k-1} \tan\theta^B_{k-1|k-1} \\ \left(\tilde{q}^B_k \cos\phi^B_{k-1|k-1} - \tilde{r}^B_k \sin\phi^B_{k-1|k-1}\right) \\ \left((\tilde{q}^B_k \sin\phi^B_{k-1|k-1} + \tilde{r}^B_k \cos\phi^B_{k-1|k-1}) \sec\theta^B_{k-1|k-1}\right) \end{bmatrix} T_s \tag{1}$$

The nine-state EKF builds from the attitude estimate. Both velocity and position are estimated through straight-forward integration of the 3-axis accelerometer data. The attitude estimate is used to transform the acceleration along the aircraft body axes (ϕ, θ, ψ) to acceleration in the local Cartesian plane (*x*, y, z). The integration of the accelerometer inputs to determine the velocity estimates (V_x, V_y, V_z) is shown in Equation 2 [18].

$$\begin{bmatrix} V^L_{x\,k|k-1} \\ V^L_{y\,k|k-1} \\ V^L_{z\,k|k-1} \end{bmatrix} = \begin{bmatrix} V^L_{x\,k-1|k-1} \\ V^L_{y\,k-1|k-1} \\ V^L_{z\,k-1|k-1} \end{bmatrix} + DCM(\phi^B_{k-1|k-1}, \theta^B_{k-1|k-1}, \psi^B_{k-1|k-1}) \begin{bmatrix} \tilde{a}^B_{x\,k} \\ \tilde{a}^B_{y\,k} \\ \tilde{a}^B_{z\,k} \end{bmatrix} T_s + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} T_s \tag{2}$$

The Direction Cosine Matrix, used to rotate the rover attitude Euler angles such that the rover is aligned with the Cartesian axes, is shown in Equation 3. The trigonometric functions *cosine* and *sine* are abbreviated as *c* and *s*, respectively.

$$DCM(\phi,\theta,\psi)=\begin{bmatrix} c\psi\, c\theta & -s\psi\, c\phi + c\psi\, s\theta\, s\phi & s\psi\, s\phi + c\psi\, s\theta\, c\phi \\ s\psi\, c\theta & c\psi\, c\phi + s\psi\, s\theta\, s\phi & -c\psi\, s\phi + s\psi\, s\theta\, c\phi \\ -s\theta & c\theta\, s\phi & c\theta\, c\phi \end{bmatrix} \tag{3}$$

The simple integration of the velocity in the Cartesian plane is given by Equation 4 [18].

$$\begin{bmatrix} x^L_{k|k-1} \\ y^L_{k|k-1} \\ z^L_{k|k-1} \end{bmatrix} = \begin{bmatrix} x^L_{k-1|k-1} \\ y^L_{k-1|k-1} \\ z^L_{k-1|k-1} \end{bmatrix} + \begin{bmatrix} V^L_{x\,k-1|k-1} \\ V^L_{y\,k-1|k-1} \\ V^L_{z\,k-1|k-1} \end{bmatrix} T_s \tag{4}$$

The velocity and position estimates are calibrated against the position and velocities provided by the incorporated GPS. This system made the trade-off between attitude accuracy and computational requirements, only uses the gyroscope-estimated attitude, and does not calibrate these estimates with the accelerometer inputs, as described in Kim's work [16]. Also of note is that the yaw is relative to the initial yaw at system initialization and does not incorporate a magnetic-north-relative heading.

Li and Wang developed an attitude estimation system that incorporated a 3-axis magnetometer, as well as 3-axis gyroscope and 3-axis accelerometer to better estimate yaw [19]. The 3-axis magnetometer, much like the accelerometer, experiences bounded error and no long-term drift. This error characteristic makes it a good candidate to correct the drift in the gyroscopes in a Kalman Filter. This approach is better for land-navigation where compass heading is important in planning a trajectory from the current point to a desired point.

Chapter 3
Rover Design, Development, and Verification

The rover design and development, a collaborative effort that was undertaken by the WVU Mountaineers Robotics Team in response to the 2012 NASA/NIA RASC-AL RoboOps challenge, is detailed in Section 3.1. Section 3.2 details the verification tests carried out to ensure that all rover systems were operational and met requirements.

3.1. Rover Design and Development

The rover is a complex system consisting of mechanical, electrical, and software systems. Three primary systems were identified which comprised the robot WVU designed: drive, sample acquisition, and control and communications (C^2). The drive system is comprised of the basic mechanical and electromechanical components necessary for rover traversal. The sensory subsystem components include a pan-tilt-zoom camera, fixed forward-looking camera, and NAS. The power subsystem includes lithium polymer (LiPo) batteries, an intelligent voltage regulator, and circuit protection components. WVU's rover can be seen in Figure 3-1, key physical dimensions are detailed in Table 3-1.

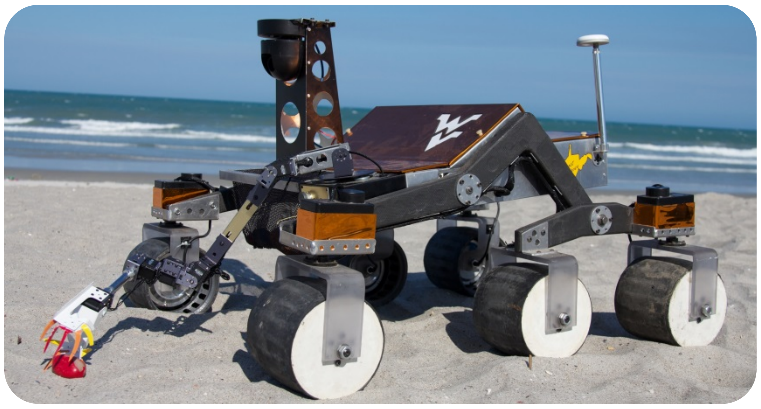


Figure 3-1: WVU Mountaineers Mars Rover

Table 3-1: Rover Physical Dimensions

Dimension	Value
Length	100 cm
Width	92 cm
Height (Stowed)	50 cm
Height (Deployed)	80 cm
Mass	45 kg

3.1.1. Drive System

The drive system consists of the basic mechanical and electromechanical components necessary for rover traversal. These include the chassis, sensory subsystem, and power subsystem components. The chassis includes the frame and drive elements such as motors, gearboxes, and wheels. The sensory subsystem consists of cameras mounted on the rover. The power subsystem consists of batteries and hardware that monitors their voltage and prevents undervoltage errors. The components of the drive system are described in Sections 3.1.1.1 - 3.1.1.3.

3.1.1.1. Chassis

The rover employs a rocker-bogie suspension system with six independently driven wheels as the foundation of its drive system. The rover is built upon a chassis featuring a rocker-bogie suspension system. The chassis has six individually driven wheels and four-wheel steering. This system does not use springs or pressurized elements, such as hydraulics or pneumatics, making the chassis more viable in most environments encountered during space exploration. The system is designed to evenly distribute the rover’s mass across all six wheels minimizing sinking in soft and uneven ground [2].

The rocker-bogie system is comprised of differential, rocker, and bogie arms. The differential is custom-built using miter gears that transmit motion using a one-to-one gear ratio, the gears are assembled so that the two sides of the differential have opposite hands of rotations, similar to the rocker-bogie systems used on NASA rovers *Spirit* and *Opportunity* [2]. The differential is built to allow ample range of movement between the rocker arms on either side of the chassis, allowing one side to traverse over ten centimeter obstacles while the other side can stay fixed on the ground. The rocker arms are joined to the differential by aluminum torque tubes. The rockers and bogies are both constructed from carbon fiber reinforced plywood, providing a strong, lightweight foundation. Figure 3-2 shows the design of both the rocker and bogie components of the rocker-bogie arms. The arms house mounts for the steering servos and wheel assemblies. The assemblies have been verified, through Finite Element Analysis (FEA) modeling, to safely handle the rover load. A sample FEA model to analyze combined top and side load on the wheel mount is shown in Figure 3-3.

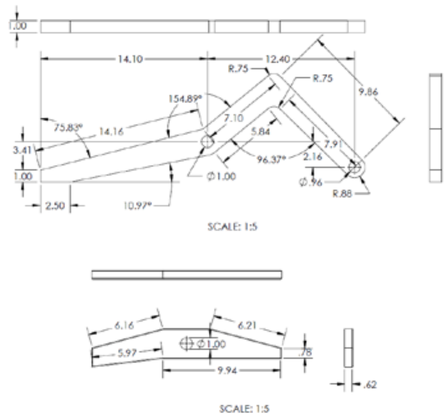
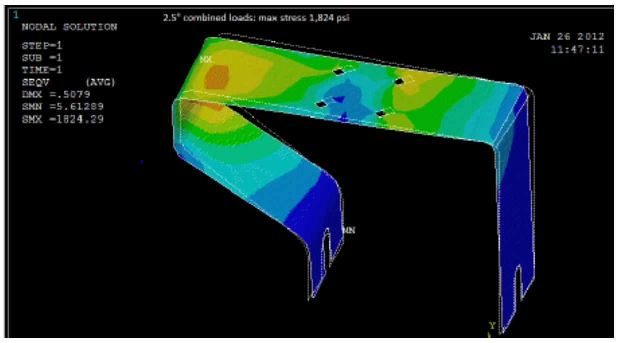


Figure 3-2: Rocker-Bogie Mechanical Drawing


```
1
NODAL SOLUTION

STEP=1
SUB =1
TIME=1
SEQV       (AVG)
DMX =.5079
SMN =5.61289
SMX =1824.29
```
2.5° combined loads: max stress 1,824 psi

JAN 26 2012
11:47:11

Figure 3-3: FEA Model of Wheel Mount

The aluminum torque tubes and differential are housed in a Fiberglass Reinforced Plastic (FRP) square channel for rigidity. A rendering of the three-dimensional model of the rocker-bogie arms, differential, and differential enclosure is shown in Figure 3-4.

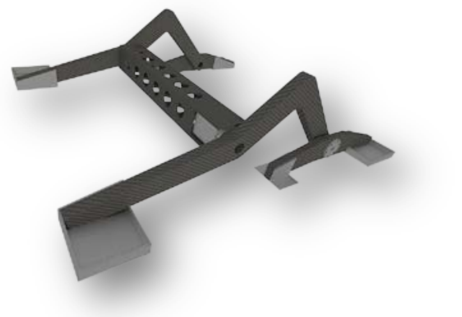


Figure 3-4: 3D Model of WVU Rover Rocker Bogie Suspension System

This channel is installed in an electronics box constructed of lightweight aluminum enclosed by an aluminum cover that creates an external platform. The box is sealed from weather and includes filtered air ports for electronics cooling. The interior of the box houses the control and communications system, the power system, and the NAS. The exterior platform houses the cameras, sample acquisition system, antennas for the wireless broadband modem and NAS, emergency stop, and battery disconnect switch. The rover peripherals are mounted in and on the electronics box such that the box's center of mass is near the pivot point of the rocker-bogie assembly. These design considerations reflect inspiration from the design of the NASA rovers *Spirit* and *Opportunity* [2].

Propulsion is provided by brushless DC hub motors, commonly used in electric bicycles. They have been modified to allow mounting inside a custom-machined PVC wheel, shown in Figure 3-5, and to provide reverse operation. The wheel motors are controlled by a RoboteQ HBL-2350 Motor Controller [20]. The wheel diameter is twenty centimeters providing ample ground clearance to negotiate ten centimeter tall obstacles per system requirements. Powerful Torxis i00600 servo mechanisms steer the rover, using position-based control provided by a Phidgets PhidgetServo 1001 4-Motor controller [21]. The servo mechanisms control the four corner-mounted wheels. This four-wheel steering design allows a near-zero turning radius, allowing the rover to effectively maneuver around obstacles. The servo, mount, wheel, and wheel bracket are shown in Figure 3-6.

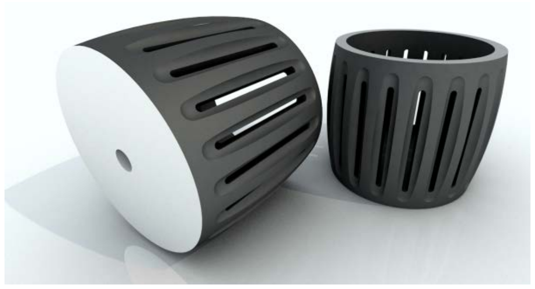


Figure 3-5: 3D Model of 20 cm PVC Wheels

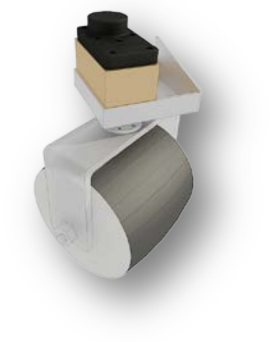


Figure 3-6: 3D Model of Steering Subsystem

3.1.1.2. Sensory System

An Axis 215 PTZ camera [22] provides the abilities to identify samples of various colors and navigable paths through the mission environment. The Axis camera was chosen because it provides both control and video over a single Ethernet interface. It features an HTTP application programmer's interface (API) providing advanced features such as switchable resolution modes and streaming video.

A second Logitech web camera, fitted with a wide-angle lens, is mounted to the front of the rover. This provides an obstacle-view camera near ground level and provides video of the rover traversal.

3.1.1.3. Power System

The power system is designed to provide ample power for at least sixty minutes and isolate the power used for the motors from the C^2 components. It includes over-current protection for the control electronics and includes an external battery cutoff switch to allow for quick disconnect of power if necessary.

The power system is comprised of the batteries, motor controllers, DC-DC voltage converter, circuit protection, and emergency battery disconnect circuit. Three LiPo batteries are required for operation. The LiPo batteries were chosen for their high capacity-to-weight ratio and their high discharge capabilities, allowing large currents to be drawn for short periods of time in the event of a motor stalling. A 5 amp-hour (Ah), 37 volt DC (VDC) battery was chosen to power the three RoboteQ HBL-2350 motor controllers. A 5 Ah 11VDC battery powers the steering servos and sample acquisition arm's Robotis AX-18A [23] servos. An 8 Ah capacity, 18 VDC battery supplies a Mini-Box DC-DC voltage converter [24] to provide a constant voltage source to the Robot Control Unit (RCU) computer and other control electronics. This separate battery is required to help alleviate any inducted noise or a large current draw from the motors affecting the control or communications hardware. Figure 3-7 illustrates the major hardware components, major sensory interfaces, and the individual power legs.

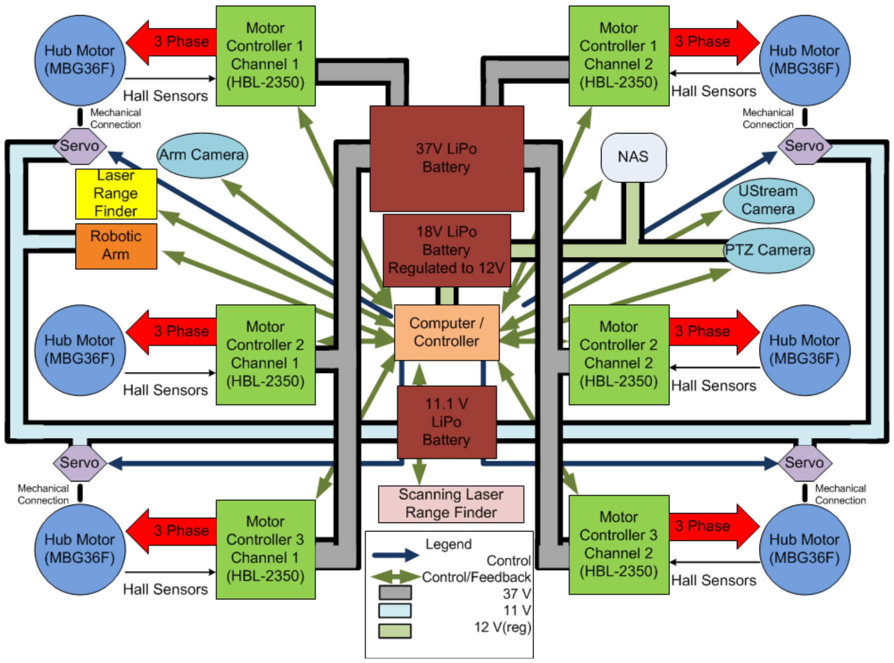

Hub Motor (MBG36F)
3 Phase
Motor Controller 1 Channel 1 (HBL-2350)
Hall Sensors
Motor Controller 1 Channel 2 (HBL-2350)
Mechanical Connection
Servo
Arm Camera
Laser Range Finder
Robotic Arm
37V LiPo Battery
18V LiPo Battery Regulated to 12V
NAS
UStream Camera
PTZ Camera
Computer / Controller
Motor Controller 2 Channel 1 (HBL-2350)
Motor Controller 2 Channel 2 (HBL-2350)
11.1 V LiPo Battery
Scanning Laser Range Finder
Motor Controller 3 Channel 1 (HBL-2350)
Motor Controller 3 Channel 2 (HBL-2350)
Legend
Control
Control/Feedback
37 V
11 V
12 V(reg)

Figure 3-7: Rover System Block Diagram

The power system also addresses two major risks. First, LiPo batteries carry a risk of damage if over-discharged. This risk is mitigated through the use of hardware with voltage sensing capability. The RoboteQ motor controllers [20], Mini-Box DC-DC regulator [24], and Dynamixel AX-18A Servo controller [23] all offer the ability to shut down when the battery voltage reaches a programmable minimum, preventing over-discharge. Finally, in the event of a loss of control or catastrophic electrical fault, power to all systems can be disconnected through the press of a single emergency stop button. The negative leads from each battery share a common ground bus that is wired through an exterior battery disconnect switch.

3.1.2. Sample Acquisition System

The sample acquisition system is comprised of a robotic arm with a custom end effector, collection bin, and control software. The robotic arm is mounted at the front of the rover platform facing forward in order to collect samples sitting in front of the rover. The control software comprises RCU code and servo controller firmware. The RCU code translates operator control inputs to servo positions which are packetized and transmitted to the servo controller. The servo controller monitors the state of each servo including torque, temperature, and position. A rendering of the three dimensional model of the sample acquisition system is shown in Figure 3-8. A photo of the sample acquisition system with the arm stowed is shown in Figure 3-9 and the arm is shown collecting a sample rock in Figure 3-10.

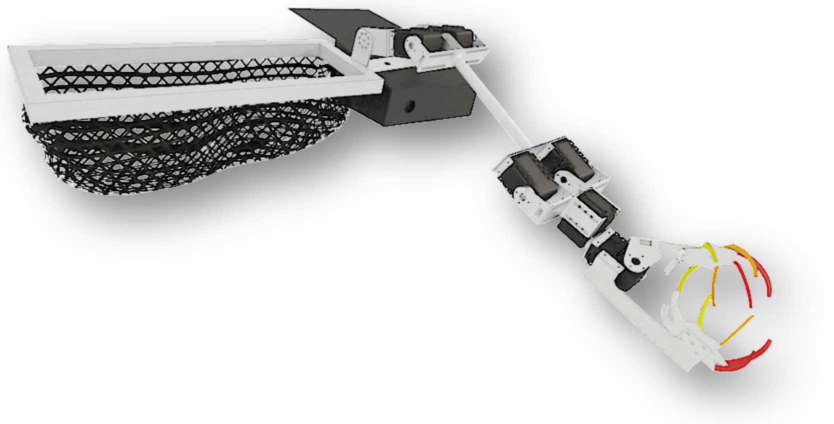


Figure 3-8: Sample Acquisition System 3D Rendering

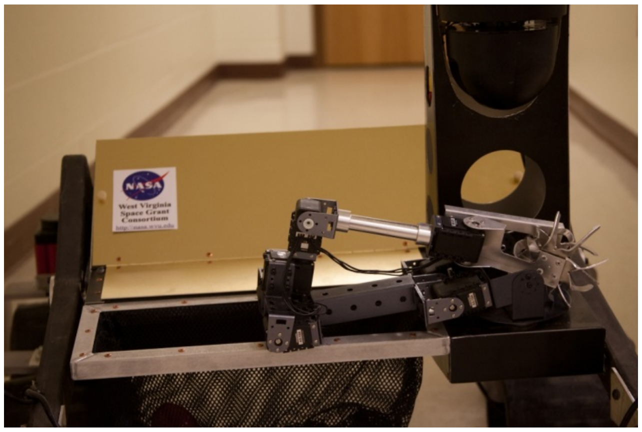


Figure 3-9: Stowed Arm

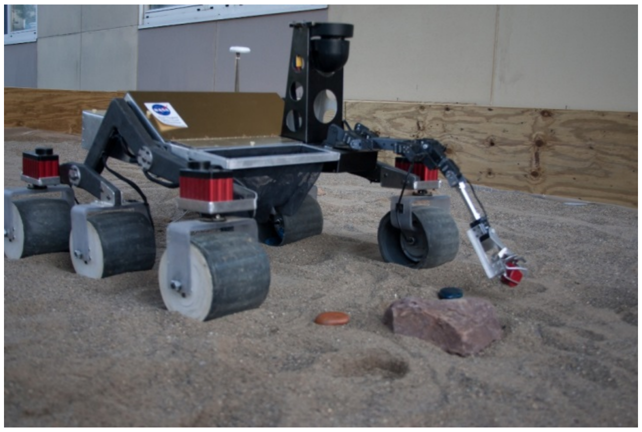


Figure 3-10: Arm collecting sample

3.1.2.1. Arm

The robotic gripper arm used is a COTS arm from Crustcrawler Robotics [25]. This arm has a lifting capacity of 900 grams and a maximum gripper opening of 14 centimeters, which exceeds system requirements. It employs eight Robotis Dynamixel AX-18A servomechanisms providing 5 degrees of freedom. Each servo provides position, load, voltage, temperature, and movement status feedback. The servos are position controlled and offer a three hundred degree range of motion, with a resolution of 0.29 degrees, and a repeatability of 2.5 millimeters [23]. This allows for precise positioning of the gripper in order to acquire samples. The forearm section was lengthened to extend the arm's reach. The arm's ability to lift masses in excess of eight hundred grams was maintained with this modification. The end-effector was modified to operate like an industrial scrap handler, using interleaved *fingers* that can cut through material around the sample and collect the sample in a basket-like structure. The fingers are arranged to hold irregular shaped items of a diameter of 1.5 centimeters to 9 centimeters when closed. This end-effector reduces the torque necessary to grasp the sample and maintains better control of the sample throughout the range of motion between the surface and the collection hopper.

3.1.2.2. Collection Bin

A collection bin is incorporated on the front of the rover for storing retrieved samples. The collection bin is constructed of high-strength nylon mesh on a lightweight aluminum frame to allow expansion and suspension of the samples while allowing smaller particles to escape so that valuable volume is not wasted. The volume, before expansion, of the collection bin is 10,000 cubic centimeters. This design allows ample room to collect thirty samples of 8 centimeters in diameter.

3.1.2.3. Control Software

The control software for the robotic arm provides both manual and automated operation. Manual operation is performed using an Xbox 360 gaming controller. The goal of automated operation mode is to reduce the level of dexterity required to operate the arm by automating as much of the sample collection process as possible, such as depositing samples in the collection bin.

3.1.3. Control and Communication Systems

The C^2 subsystem includes the hardware and software necessary to control and communicate with the drive and sample acquisition systems. The C^2 sub system includes the RCU, motor controllers, arm servo controller, Operator Control Unit (OCU), and network hardware. The system has been tested with both a wireless cellular modem and WiFi network.

The rover is controlled by two computers: the OCU and RCU. The OCU is housed on a server at WVU. It displays video and telemetry from the rover and serializes operator input to be transmitted to the rover. The RCU resides on the rover and provides four primary functions: receiving and parsing control state data from the OCU, executing algorithms for autonomous operation macros, interfacing to the attached devices and controllers, and relaying video and telemetry to the OCU. The two control units are linked by a communications system incorporating a wireless broadband modem. Figure 3-11 shows the communications relationship between the OCU and RCU.

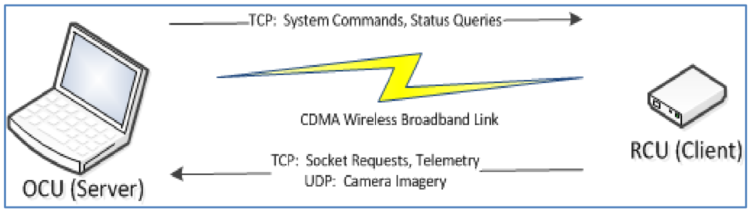

TCP: System Commands, Status Queries
CDMA Wireless Broadband Link
TCP: Socket Requests, Telemetry
UDP: Camera Imagery
OCU (Server)
RCU (Client)

Figure 3-11: Communications Relationship Diagram

3.1.3.1. Control Hardware

The control hardware comprises the RCU and motor controllers, providing control on-board the rover. The RCU is discussed in Section 3.1.3.1.1. The motor controllers for drive motors and steering servos are discussed in Section 3.1.3.1.2. It also includes the OCU hardware including network and computer hardware at WVU as well as the gamepad controller interface discussed in Section 3.1.3.1.3.

3.1.3.1.1 RCU Computer

The ‘brain’ of the RCU is the onboard computer. It receives and dispatches operator commands, interfaces to the robot's subsystems, processes video from all cameras, and performs autonomous macros to aid the operators in driving the robot, navigating the rock yard, and operating the sample acquisition arm.

The components for the RCU computer were chosen for reliability and performance. The CPU selected is a dual-core 1.86 GHz Intel Atom N2800 low-power-consumption processor. The N2800 has a low thermal design power (TDP) of 8W and also provides significant computing capability with a 1.86 GHz clock speed [26]. The processor is mounted in a passively cooled industrial-grade Intel DN2800MT Marshalltown Mini-ITX motherboard. The board supports 2 RS-232 ports on integrated headers [27] and four more through the use of a Jetway 4-port RS-232 Mini-PCIe module. While the motor controllers, arm controller and NAS support both serial and USB connections, RS-232 has better recovery in the presence of electrical issues [28]. A Solid-State Drive (SSD) serves as the primary storage. The absence of moving parts makes the SSD more resilient to vibration and impact than hard disc drives [29].

3.1.3.1.2 Motor Controllers

Three RoboteQ HBL-2350 2-channel brushless DC motor controllers were chosen to control the rover’s six wheel motors. These advanced motor controllers provide open or closed-loop motor control utilizing the Hall-effect sensors in each wheel [20]. Each wheel is individually powered allowing a level of redundancy; if one wheel or channel fails, the other five can continue to propel the rover in the degraded state. This is similar to what happened to the NASA *Spirit* when one of its wheels failed in 2006 [3], yet it was able to continue its mission for over three years after the failure [30].

A Phidgets 1001 PhidgetServo 4-motor controller provides pulse-width-modulated (PWM) position output to the four Torxis i00600 servos, mounted at the four corners of the rover for steering. The 1001 controller provides step accuracy of 0.1 degree, has a wide-range voltage input of 6-12 volts DC, and communicates to the RCU computer via Universal Serial Bus (USB) [21]. Furthermore, the 1001 controller is small and light, measuring approximately 36x40millimeters, with a mass of 45 grams [21].

3.1.3.1.3 OCU Control Hardware

The hardware for the OCU is comprised of three components: a server on the WVU network and two USB-connected Xbox 360 controllers. The server has a graphics card capable of running two monitors to give the rover and arm operators their own displays.

3.1.3.2. Control and Communications Software

The communications system is crucial for proper teleoperation of the rover from the remote site on WVU's campus. The requirements dictate that the rover utilize a Verizon wireless modem, however Verizon Wireless does not offer 3G or 4G wireless service in the vicinity of WVU. For this reason, 3G wireless testing is being performed with a US Cellular PCD UM185 3G wireless modem.

The US Cellular wireless modem operates behind a double network address translation (NAT) layer exactly as its Verizon wireless phone counterparts, thus it is expected that the Verizon wireless modem will exhibit similar behavior. The behavior of the double NAT layer on the 3G wireless requires that the communications software on the RCU be a TCP/IP client. A TCP/IP client has the ability to "punch through" the double-NAT layer and establish a connection to the OCU server software at WVU.

Both the RCU and OCU software have been designed to be flexible, safe, and reliable. Flexibility is achieved through the use of object-oriented design. The architecture is easily adaptable to modifications of the hardware configuration, such as changes in joystick controllers, motor controllers or wiring. These changes need only be reflected in an Extensible Markup Language (XML) based configuration file, and no recompilation is necessary. The RCU software implements a multi-threaded design enabling it to be more resistant to failure and to have multiple watchdog timers to avoid erratic and/or unsafe behavior.

3.1.3.2.1 Network Communication Software

The wireless modem based network gives two options for socket-based communication: Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). TCP is a connection-oriented protocol, in which two devices communicate between each other using a handshaking procedure. The handshaking procedure enables synchronization between the devices and methodology to request re-transmission of lost packet data to prevent data loss. UDP is a connectionless protocol without any handshaking procedure. The lack of handshaking is faster, but does not provide any data loss prevention.

Any data being transmitted from the rover (RCU) to the OCU is sent using UDP for these reasons:

- It is not critical that every pack of telemetry data (GPS coordinates, compass heading, etc) reach the OCU. If a packet of telemetry gets lost, the packet will simply be replaced with more up-to-date data.
- Video camera MJPEG frames are transmitted most efficiently using UDP. For the same reason mentioned in number one, if a single frame is lost it will simply be replaced with a newer frame.
- Any unsolicited communications, such as transmitting of GPS telemetry and imagery, must be sent via UDP from the rover due to the limitations of the cellular wireless.

Rover commands from OCU to RCU, such as camera movement and rover movement, are sent via a TCP link. The RCU server is a TCP/IP client that constantly connects to the OCU TCP/IP server. Once the connection has been established commands can be transmitted directly to the RCU from the OCU. This TCP/IP connection effectively opens a “tunnel” between the OCU and RCU that allows commands to be sent directly through the wireless firewalls and NAT.

Control of the bandwidth usage is provided by user-adjustable parameters for video and telemetry. The rates at which the RCU sends telemetry and video updates, as well as the video quality, are adjustable on the fly. This dynamic capability provides maximum control over the distribution of bandwidth to allow for optimal balance depending on the network performance and the task at hand.

3.1.3.2.2 RCU Software

The RCU software is responsible for motor control, multiple camera control and image acquisition, broadcasting telemetry to the OCU, performing NAS calculations (including the EKF), and accepting commands from the OCU. The software employs multiple threads to provide optimal performance and interaction between each software component. The use of separate threads to handle operation of each individual component allows all of the components, which may operate at different update frequencies, to function harmoniously. The division of work amongst these classes and threads isolates each component from the others, ensuring that the failure of a single one does not adversely affect the rest of the system. A non-essential component could fail, but the rest of the rover can continue operation.

Safety mechanisms are implemented with watchdog timers in the device controller layer. This provides each controller its own timeout policy and enforcement procedures, and isolates the task from the communications system. The OCU and RCU maintain a heartbeat signal between them so the RCU does not continue operating under loss of the OCU. A timeout between the OCU and RCU will cause rover motion to cease by triggering the independent timeout policies for each motor controller.

Rover telemetry includes GPS/IMU fused and filtered data, digital compass data that includes roll, pitch, heading, and battery voltage levels. All telemetry software classes implement an interface titled *IProvideTelemetry*. This interface specifies the *contract* that telemetry classes must adhere to so that they can be easily integrated into the RCU software. This object-oriented mechanism is used so that any future sensors can simply implement the *IProvideTelemetry* interface and then be added to the RCU software.

Each telemetry device is implemented in its own class with its own data acquisition thread. For serial devices, the *SerialPort_DataReceived* event is used to capture all data and inserted into a queue. Then, a background worker thread is responsible for dequeuing the data and raising a

TelemetryUpdated_Event. This event is part of the *IProvideTelemetry* interface contract and is required to be implemented.

The RCU packages all telemetry into a single packet which is serialized and transmitted via UDP to the OCU. To compensate for various update rates from different sensors, the *IProvideTelemetry* interface requires that device classes (e.g., GPS, compass) provide a mechanism for the user of the class to specify how often they want telemetry data provided via the *TelemetryUpdated_Event*. The telemetry devices provide data at different rates: the NAS updates at 10Hz and the motor controllers' telemetry rates are polled at the operator-specified telemetry rate. To handle this difference in update rates and provide a means to package all telemetry into a single packet, the user of each class will specify how often they want updated telemetry. For the RASC-AL competition, it is not necessary to transmit telemetry data faster than once a second, so these devices are instructed to raise their *TelemetryUpdated_Event* once every two seconds (as an example). The background thread is always processing data thus ensuring that telemetry is not stale when reported.

Web camera and Axis camera imagery is acquired by integrating the AForge library into the RCU software. AForge is an open source set of libraries for incorporating robotic techniques such as computer vision, artificial intelligence, and image processing into robotic software [31]. This library provides a simple interface to capture individual frames from the video sources. To minimize bandwidth use, the frames captured from the cameras are compressed and transmitted in JPEG format. The level of compression and frame rate can be varied dynamically to provide the best balance for the current task.

3.1.3.2.3 OCU Control Software

The operator control unit software has two main tasks: process user input to control the state of the rover and display telemetry feedback from the RCU. The OCU is designed for two operators: the rover operator and the sample acquisition system operator. The OCU software provides each operator their own graphical user interface (GUI), allowing them to work in concert with each other to achieve mission objectives.

The rover operator can control the rover with an Xbox 360 controller, adjust network parameters (IP endpoint, communication rate), adjust controller settings (joystick sensitivity), provide, and also select individual types of telemetry (video feed, position information, battery voltages) to display to their preference. The sample acquisition operator can operate the robotic arm and laser rangefinder, set trajectories to sample locations with an Xbox 360 controller, and adjust controller settings (joystick sensitivity). A key feature of the OCU software is the ability to vary the quality and update frequency of the video feeds from each camera. Given the restricted bandwidth, balancing the quality of the images versus the frequency of transmission will be a crucial, and often-changing, element of operation. The user is able to change these parameters with minimal effort using controls built directly into the OCU software, as shown in Figure 3-12. Any changes are then forwarded to the RCU and immediately enacted. This allows the operators to improve image quality when fine detail is needed, such as when searching and acquiring samples. Conversely, the quality can be dropped and the frequency increased when a smoother image feed is desired, such as when driving the rover.

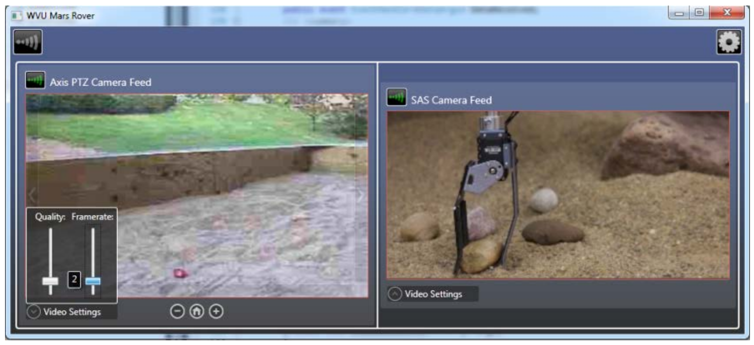


Figure 3-12: View of OCU software illustrating video quality control

3.2. Rover Testing

Tests were planned and implemented throughout the system development in order to ensure that the rover would meet all requirements of the 2012 RASC-AL RoboOps competition [1] as well as derived system requirements. These tests included isolated tests of the rover's individual subsystems including communications, the sample acquisition system, the chassis, and the power system. Finally the entire system was tested by operating the rover in mission-based scenarios. This section describes the tests performed and the observed results.

3.2.1. Communications Testing and Results

The communications framework software has been thoroughly verified to ensure that communications between the operators and the rover will be robust and reliable. During the RASC-AL RoboOps competition the rover operators must communicate with the rover using a wireless broadband modem [32]. Communications testing focused on the latency, throughput, and behavior of the wireless modem in various scenarios of rover system design. The primary data transmitted on the communications channel are TCP command packets and camera imagery. The JPEG imagery requires the largest amount of bandwidth; the TCP command packets' use of bandwidth is negligible compared to this. Therefore JPEG images, taken from the rover's PTZ camera, were sent via the communications channel and performance data were logged. In order to monitor one-way latency (RCU → OCU), a stopwatch was filmed and the difference between the time shown on the stopwatch display and the time shown in the image of the stopwatch was recorded.

- Latency for imagery was consistent at approximately one second; this was observed during a good 3G signal (3+ bars).
- Throughput testing resulted in a 0.55Mbps download and 0.35 upload rate.

The performance matched the stated performance of Verizon 3G/4G in Houston, indicating that the communications software was not injecting any further latency or decreasing throughput.

Fault protection testing was performed in addition to the performance testing of the communications channel. The possible identified faults were:

- Degraded Communications – Poor cellular reception
- Loss of OCU (Server) Communications – Loss of power, network failure, or other failure causing server to not be available to client
- Loss of RCU (Client) Communications – Loss of power, network failure, or other failure causing client to disconnect from server

Degraded communications testing has been performed to evaluate the robustness of the software. Testing conditions included very poor 3G service and Single-Carrier Radio Transmission Technology (1xRTT) service which operates at a maximum throughput of approximately 0.15 Mbps.

- Latency increased a variable amount reaching a maximum of approximately 10 seconds.
- RCU loss of communication occurred, but the RCU safed itself by stopping all motors and continually attempting to reestablish communications.

The OCU is the server in the system and is responsible for *listening* for client connection requests from the RCU.   The OCU is responsible for monitoring the health of the RCU connection, and if lost, close the socket and listen for reconnect.   This is necessary because the OCU will only accept one incoming request from the RCU to ensure there is not interference.   During OCU tests, the OCU was disconnected from its local network then reconnected.   Upon reestablishing local connectivity, it listened and reestablished communications with the RCU.

The RCU software is a TCP Client; it is responsible for always maintaining a communications link to the OCU server. The loss of RCU communications testing was performed by disconnecting the modem and by restarting the RCU.   Both tests demonstrated that if the communications fail, the RCU will repeatedly attempt to connect to the OCU automatically, and will connect when the communications channel is available.

3.2.2. Chassis Testing and Results

Chassis test design and execution was driven by system requirements, derived from the RASC-AL RoboOps competition requirements. These required the rover to be able to traverse rocks, sand, obstacles of a height up to 10 cm, and a 33% grade [32]. The rover, with full payload, was tested in the environments that met the criteria established in the system requirements. Each of the following scenarios, satisfying system performance requirement criteria, was observed through performance testing:

- The rover is able to make multiple consecutive traversals across a 3 meter long sandbox, filled with loose sand at a depth of approximately 0.5 meter. The rover is shown in the sandbox in Figure 3-13
- The rover was able to drive over a 10 cm rock without becoming stalled or being impeded
- The rover was able to ascend and descend a rocky 30% grade with no slippage or impedance.
- In addition the rover was able to ascend and descend a 54% obstacle strewn grade with some light slipping.

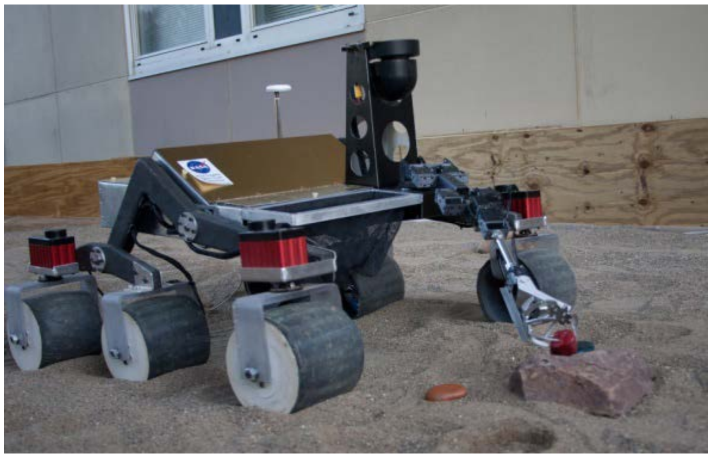


Figure 3-13: Mars Rover in Sandbox Area

3.2.3. Sample Acquisition Testing and Results

The sample acquisition system verification test design and its execution were driven by system requirements from the RASC-AL RoboOps competition requirements. The competition required rovers to be able to lift rock samples of diameters ranging from 2-8 cm and of masses ranging from 20 to 150 g [32]. In order to meet these requirements, the system requirements for the WVU rover included being able to lift items ranging in diameter from 1.5 cm to 10 cm and ranging in mass from 20 – 800 g. In addition there was a derived requirement to be able to pick up samples in sand, which was identified as a challenge in early testing. The diameter and mass range tests were carried out with a variety of rocks and a measured weight of 900g.

- The rover was able to lift rocks ranging from 1.5 - 9 cm in diameter.
- The rover was able to lift a full water bottle, with mass of 900g

In order to perform testing that closely mimicked the competition, rock samples of multiple diameters and masses were routinely placed in the sandbox and the operators, physically separated from the rover, were able to pick up all samples from the sand base. The rover collecting a rock sample from sand is illustrated in Figure 3-14.

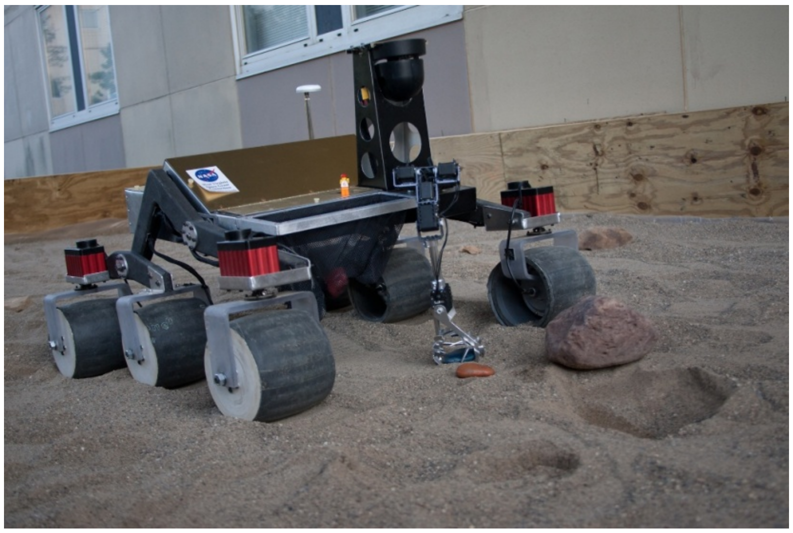


Figure 3-14: Rover Collecting Rock from Sand Surface

3.2.4. Power System Testing and Results

The power system's verification test design and its execution were driven by system requirements derived from the RASC-AL RoboOps competition requirements. The competition allows up to one hour for each rover to complete its competition run [1]. The rover was required to be able to run for 1.5 hours to provide an operational margin. The test required the operators to constantly maneuver the rover for 1.5 hours. The rover was driven for 1.5 hours at varying speeds, picked up samples, and maintained communications for the entire time, demonstrating its ability to meet the performance requirement.

Chapter 4
Navigation Assistance System Design and Development

The navigation method for the rover is teleoperation, based on imagery transmitted from the onboard cameras. This imagery provides situational awareness in the rover's near-field, and its reception at the OCU is delayed by communications latency. It was observed during the 2012 RASC-AL RoboOps competition that control of the rover, with such limited and delayed feedback, was extremely difficult. In order to maintain control and ensure the rover's safety was not compromised, the operator made small movements from the controller and then waited for the video to catch up to evaluate the safety of the next move. Furthermore, a third operator had to attempt to track the rover's movements and manually correlate them to Google Earth satellite imagery in order to estimate the rover's position in the rock yard.

In order to facilitate faster traversal between areas in the operating environment, a NAS is proposed to provide precise position and attitude data onboard the rover for telemetry and potentially performing autonomous movements. The NAS maintains the rover position and attitude input by fusing complimentary data from IMU sensors and a GPS receiver, using an EKF, in order to form an optimal estimate of the rover's position, velocity, and attitude states. Both the IMU and GPS suffer from noise that degrades the accuracy of their output. Section 4.1.1 describes the GPS' sources of error and Section 4.1.2 describes the sources of error for the IMU. The EKF helps suppress the inherent noise in the sensor systems to provide a better estimate of the states than could be achieved by using either of the systems alone.

The NAS is a composite of hardware, simple interface firmware to read and transmit sensor data, and sophisticated data processing software on-board the RCU. The NAS processes data at the GPS output rate, 10 Hz, and sends telemetry at a slower, user-configurable rate to minimize bandwidth usage. The rover attitude and position are sent back to the OCU via telemetry, providing location feedback that can be overlaid on a map at the OCU in order to help the operator navigate the mission environment.

The NAS proposed utilizes the same hardware and much of the same processing as that of Gu et al. [18]. The NAS differs from the work of Gu's team by incorporating accelerometer data to calibrate the gyroscope-predicted pitch and roll of the rover. It also derives a compass heading using the three-axis magnetometer input that is used to calibrate error in the gyroscope-predicted yaw. In addition to suppressing gyroscope noise, this creates a heading referenced to true north that can be used in overland navigation.

The hardware components used in the design are described in Section 4.1. The system design is presented in Section 4.2. The development of the EKF for data fusion of the IMU and GPS data is described in Section 4.3.

4.1. NAS Hardware Components

The NAS subsystem hardware, shown in Figure 4-1, is comprised of an Analog Device ADIS-16400 MEMS IMU [33], augmented with a NovAtel™ OEM615™ GPS [34], to provide traversal information. A NetBurner® Mod5213 microcontroller [35] interfaces to each sensor and combines the data from both sensors into a single binary data packet and transmits it on the RS-232 data port at a frequency of 10 Hz.

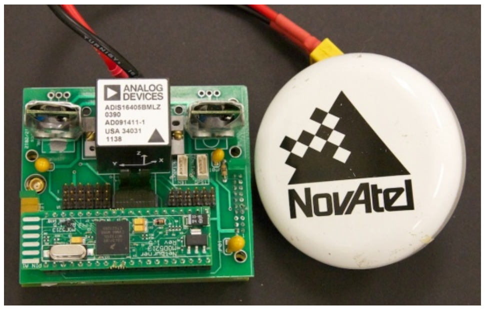


Figure 4-1: Navigation Assistance System

The NovAtel GPS receiver is described in Section 4.1.1. The Analog Device's IMU is described in Section 4.1.2. The NetBurner Mod5213 microcontroller that interfaces between the IMU, GPS and the RCU computer, is described in Section 4.1.3.

4.1.1. GPS Receiver

The GPS receiver selected for the NAS is the NovAtel OEM615. The board, shown in Figure 4-2, was selected due to its high quality and small, lightweight package. As configured, the GPS unit uses a single antenna receiving L1 frequency signals and is capable of up to 1.5 meter accuracy [34]. The single antenna, shown in Figure 4-3, used is a NovAtel ANT-26C1GA-TBW-N. This antenna features an internal 33 dB amplifier (powered by the OEM 615 receiver) and small, lightweight form-factor [36]. The GPS is configured to output the Earth Centered-Earth Fixed (ECEF) location of its antenna at a rate of 10Hz.

Figure 4-2: NovaTel OEM615 GPS Board [34]

Figure 4-3: NovaTel GPS Antenna [36]

The GPS, while capable of providing a set of coordinates within 1.5m of the actual location of the receiver, can be degraded by a variety of phenomena [13]. Atmospheric interference can speed up or slow down signals transmitted from GPS satellites, causing ranging errors. Receivers on the ground can suffer from multipath interference, where the GPS signals reflect of near items such as trees or buildings. If the satellites that are in view have a poor aperture for triangulation algorithms, this too can inject error into the system.

4.1.2. Inertial Measurement Unit

The Analog Device's ADIS16400 IMU provides inertial data to the NAS. It incorporates a three-axis gyroscope, three-axis accelerometer, three-axis magnetometer, and temperature sensor. It also includes a suite of configurable analog and digital inputs/outputs that are not used by this project. The

Micro-Electro-Mechanical System (MEMS) sensor provides this functionality in a package that is approximately 23mm^3 and has a weight of approximately 16g [33]. This makes it ideal for applications where volume and mass budgets are constricted. The sensor communicates via the Serial Peripheral Interface (SPI) bus, providing synchronous communications at a serial clock rate up to 2MHz [33]. In order to minimize processing on-board and communications bandwidth, the outputs are encoded as fourteen bit, two's complement integers that are multiplied by floating point resolutions, provided in [33], in order to determine the output in floating-point precision scientific units.

Figure 4-4 shows the axial orientations for the sensors, and is described as follows:

- The arrows a_x, a_y, and a_z show the direction of acceleration that produce a positive output
- The arrows m_x, m_y, and m_z show the direction of a magnetic field that produce a positive output
- The arrows g_x, g_y, and g_z show the direction of rotation that produces a positive output

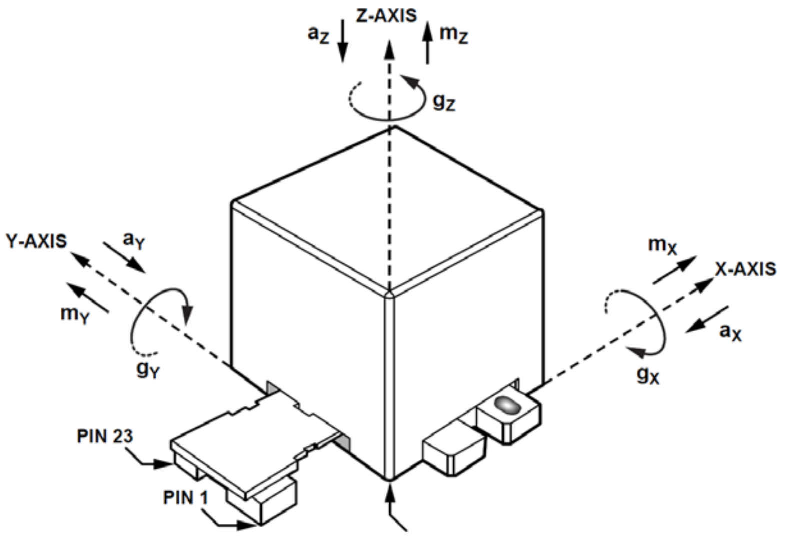


Figure 4-4: ADIS16400 Axial Orientation Diagram [33]

The accelerometer and gyroscope data are both rates, the acceleration vector provided by the accelerometers is the rate of change of velocity, and the gyroscopes provide the angular rate of change about the three axes. This data suffers from bias error, wherein the IMU reports acceleration and angle-rate data when there actually is no acceleration or angular movement [37]. In addition, these sensors suffer from bias instability; the bias error has a random variation. The sensor data can drift over time, because the sensor outputs must be integrated in order to determine attitude and velocity, small errors can propagate quickly and cause the reported state to diverge from the actual state.

4.1.3. GPS/IMU Serial Interface

The interface between the GPS, IMU, and RCU computer is implemented using a NetBurner MOD5213 Microcontroller. The MOD 5213 Microcontroller, shown in Figure 4-5, houses firmware that receives GPS packets at a rate of 10 Hz and, upon receipt of a GPS packet, polls the IMU for data. The MOD5213 provides an SPI bus and up to three Universal Asynchronous Receiver/Transmitters (UARTs) [35]. The MOD5213 SPI bus runs at 0.258 MHz and controls the interface to the IMU SPI bus. Two UARTs are used in this system. The first provides communications with the GPS at 3.3V CMOS logic levels. The second provides packetized data to the RCU. This interface is implemented by shifting the MOD5213 3.3V CMOS signal levels to RS-232 levels, utilizing a level shifter Integrated Circuit (IC) chip.

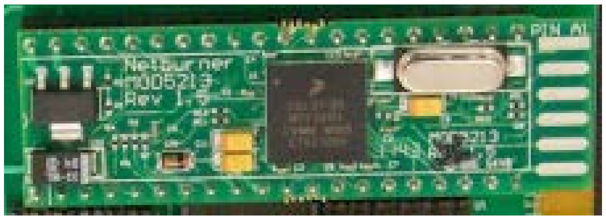


Figure 4-5: NetBurner MOD5213

4.2. NAS System Design

The components, listed in Section 4.1, are combined and integrated with the RCU control computer to form the NAS subsystem.

Figure 4-6 shows the block diagram of the NAS subsystem.

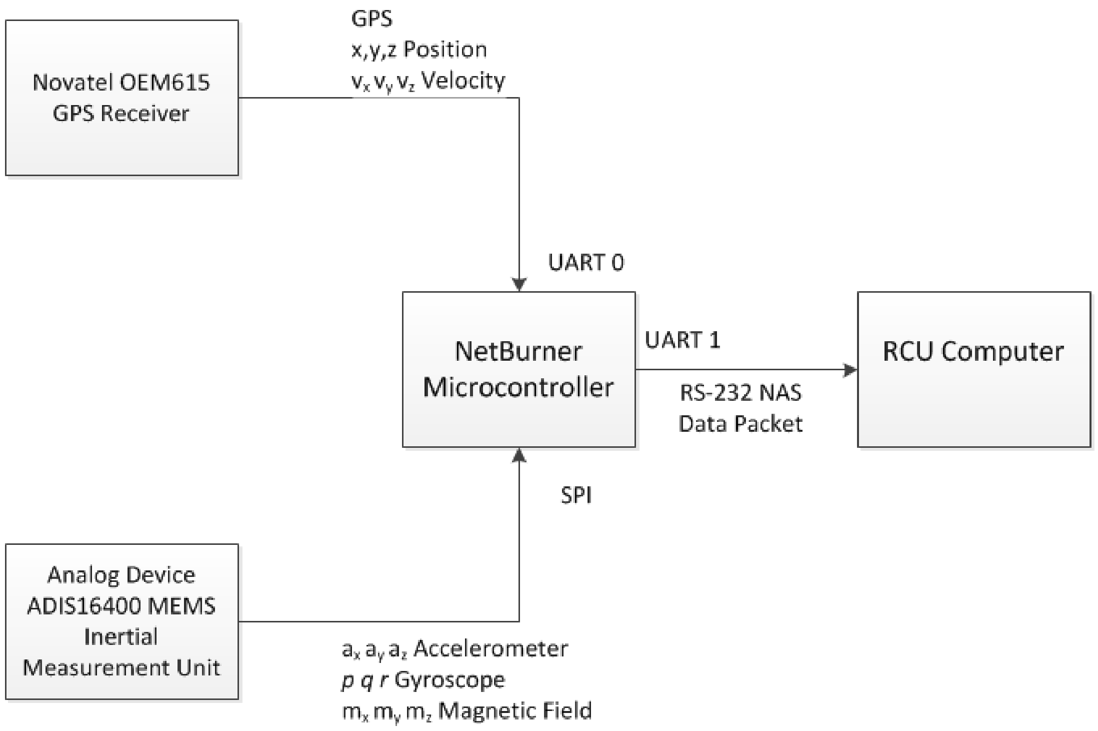

Novatel OEM615 GPS Receiver
GPS
x,y,z Position
v_x v_y v_z Velocity
UART 0
NetBurner Microcontroller
UART 1
RS-232 NAS Data Packet
RCU Computer
SPI
Analog Device ADIS16400 MEMS Inertial Measurement Unit
a_x a_y a_z Accelerometer
p q r Gyroscope
m_x m_y m_z Magnetic Field

Figure 4-6: NAS Block Diagram

The NetBurner combines IMU data, transmitted as a two's complement fourteen bit number (in two bytes), with GPS data, transmitted in Earth-Centered-Earth-Fixed (ECEF) format as a set of 64-bit double precision floating point numbers. The data is transmitted to the RCU computer in the packet format described in Table 4-1.

Table 4-1: NAS Output Packet Data

Data Description	Resolution	Range
Accelerometer X	3.33m*g* / LSB	±18 *g*
Accelerometer Y	3.33m*g* / LSB	±18 *g*
Accelerometer Z	3.33m*g* / LSB	±18 *g*
Gyroscope X	0.05° / sec / LSB	±350°/sec
Gyroscope Y	0.05° / sec / LSB	±350°/sec
Gyroscope Z	0.05° / sec / LSB	±350°/sec
Magnetometer X	0.5 milligauss / LSB	±3.5 gauss
Magnetometer Y	0.5 milligauss / LSB	±3.5 gauss
Magnetometer Z	0.5 milligauss / LSB	±3.5 gauss
Temperature	0.14°C / LSB	-
GPS ECEF Position X	meters (64-bit Double)	
GPS ECEF Position Y	meters (64-bit Double)	
GPS ECEF Position Z	meters (64-bit Double)	
GPS ECEF Velocity X	meters / sec (64-bit Double)	
GPS ECEF Velocity Y	meters / sec (64-bit Double)	
GPS ECEF Velocity Z	meters / sec (64-bit Double)	

The RCU performs the preprocessing of the data prior to providing it to the EKF. This includes:

- Converting the IMU data to engineering units by multiplying by the resolution shown in Table 4-1
- Converting the ECEF GPS coordinates to the local East North Down (END) Cartesian coordinate plane
- Determining accelerometer-based roll and pitch
- Determining tilt-compensated compass heading utilizing the roll and pitch estimates from the previous EKF state and the magnetometers

The RCU then performs data fusion using a nine-state EKF. The EKF predicts the estimates of three-dimensional position (x, y, and z) and velocity (v_x, v_y, and v_z) using the IMU's accelerometer outputs, and predicts the three attitude axes (roll, pitch, and yaw) using the gyroscope outputs. The predicted position and velocity estimates are then calibrated with measurements from the GPS, and the attitude estimates are calibrated with the accelerometer and magnetometer outputs. This calibrated state information is provided as telemetry to the OCU and to the AutoNavigation algorithm, if enabled. The EKF state estimation is described in more detail in Section 4.3.

4.3. GPS/IMU Data Fusion

The system errors inherent in both IMU and GPS systems can degrade the output from a navigation system that is based solely on either one. However, by using complimentary data from each system an optimal state estimate can be derived using an EKF. The EKF considers the uncertainty in each system and provides a state estimate output that is a ratio of the output from each system. The IMU is a better system in the short-term and is used to *predict* the state. The IMU's accelerometers more stable, bounded-error, output of attitude, however these can suffer from drift error over time. The GPS provides a more stable, bounded-error, output of velocity and position. These outputs are used as the *measurement* to calibrate the estimate in the EKF algorithm.

The nine states estimated for the rover are the three-axis position (x, y, z) velocity (V_x, V_y, V_z) and attitude estimates (roll, pitch, heading), similar to the work of Gu et al [18]. This work differs from that of Gu et al., in that their system does not calibrate gyroscope-estimated roll and pitch with accelerometer data and it determines yaw and not heading; yaw being the value of rotation about the z-axis from the initial attitude and heading being a compass direction referenced to true north. The roll and pitch is much more dynamic for an aircraft versus a ground-navigating rover. Small noise in the gyroscope-estimated roll and pitch is more prevalent in a ground-based system, as small errors can accumulate through integration over time. This project calibrates the gyroscope-estimated roll and pitch with the roll and pitch determined from the accelerometers. In addition to minimizing the integration error for the gyroscope-derived attitude estimates, it also eliminates the need for *a priori* attitude information to determine the correct attitude with respect to level. These values are important for tilt-compensating the magnetometer to determine magnetic heading. The tilt-compensated heading is used to calibrate the gyroscope-estimated *yaw* in the EKF. This process minimizes error in the heading estimation, as well as places it in the reference of true north. The data fusion algorithm is broken down as follows. Section 4.3.1 details the prediction of the nine rover states tracked by the EKF, including equations for translating sensor input into position, velocity, and attitude estimates. Section 4.3.2 details the measurement inputs, including coordinate transformations for GPS positions and velocities, roll and pitch estimation from accelerometer inputs, and tilt compensated *heading* estimation from magnetometer and attitude input. Section 4.3.3 details the EKF development including algorithm additions to the basic methods described in Section 2.3 and tuning the noise covariance matrices.

4.3.1. Rover State Prediction Process

The first step of the EKF is to determine the three-axis position (*x, y, z*) velocity (V_x, V_y, V_z) and attitude estimates (roll, pitch, yaw) by integrating the outputs from the IMU's gyroscopes and accelerometers. These estimates are highly responsive to short-term movements, but errors can grow unbounded through the process of integration. This section details the prediction of the nine states through integration and recursive use of the EKF's prior state to bound errors introduced through integration, while maintaining the responsiveness of the sensors.

The attitude estimate is predicted by straight-forward integration of the roll-rates from the three-axis gyro. Equation 5 shows the relationship of the predicted roll (ϕ), pitch (θ), and heading (ψ) with the EKF's previous attitude state and gyroscope inputs (p, q, r) [18].

Note the tilde above the three gyroscope inputs; this denotes the addition of a random measurement noise term. The dash superscript denotes a prediction, and the hat accent denotes an estimate.

$$\begin{bmatrix}\hat{\phi}_k^- \\ \hat{\theta}_k^- \\ \hat{\psi}_k^-\end{bmatrix} = \begin{bmatrix}\hat{\phi}_{k-1} \\ \hat{\theta}_{k-1} \\ \hat{\psi}_{k-1}\end{bmatrix} + \begin{bmatrix}\tilde{p}_k + \tilde{q}_k \sin\phi_{k-1} \tan\theta_{k-1} + \tilde{r}_k \cos\phi_{k-1} \tan\theta_{k-1} \\ \tilde{q}_k \cos\phi_{k-1} - \tilde{r}_k \sin\phi_{k-1} \\ (\tilde{q}_k \sin\phi_{k-1} + \tilde{r}_k \cos\phi_{k-1}) \sec\theta_{k-1}\end{bmatrix} T_s \tag{5}$$

Where:

$\hat{\phi}_k^-$	*= Estimated roll prediction at time k*
$\hat{\theta}_k^-$	*= Estimated Pitch prediction at time k*
$\hat{\psi}_k^-$	*= Estimated Heading prediction at time k*
ϕ_{k-1}	*= Estimated roll at time k-1 (EKF output)*
θ_{k-1}	*= Estimated pitch at time k-1 (EKF output)*
ψ_{k-1}	*= Estimated heading at time k-1 (EKF output)*
$\tilde{p}_k$	*= Rotation about x-axis + sampling noise*
$\tilde{q}_k$	*= Rotation about y-axis + sampling noise*
$\tilde{r}_k$	*= Rotation about z-axis + sampling noise*
T_s	*= Sampling Period*

The nine-state EKF builds from the attitude estimate. Both velocity and position are estimated through straight-forward integration of the three-axis accelerometer data. The attitude estimate is used to transform the acceleration along the rover body axes (ϕ, θ, ψ) to acceleration in the local Cartesian plane (x, y, z). The integration of the accelerometer inputs to determine the velocity estimate predictions (V_x, V_y, V_z) is shown in Equation 2 [18].

Note the tilde above the three accelerometer inputs; this denotes the addition of a random measurement noise term. The dash superscript denotes a prediction, and the hat accent denotes an estimate.

$$\begin{bmatrix} \hat{V}_{x\ k}^- \\ \hat{V}_{y\ k}^- \\ \hat{V}_{z\ k}^- \end{bmatrix} = \begin{bmatrix} \hat{V}_{x\ k-1} \\ \hat{V}_{y\ k-1} \\ \hat{V}_{z\ k-1} \end{bmatrix} + DCM(\hat{\phi}_{k-1}, \hat{\theta}_{k-1}, \hat{\psi}_{k-1}) \begin{bmatrix} \tilde{a}_{x\ k} \\ \tilde{a}_{y\ k} \\ \tilde{a}_{z\ k} \end{bmatrix} T_s + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} T_s \tag{6}$$

Where:

$\hat{V}_{x\ k}^-$	*= Estimated velocity prediction along x axis at time k*
$\hat{V}_{y\ k}^-$	*= Estimated velocity prediction along y axis at time k*
$\hat{V}_{z\ k}^-$	*= Estimated velocity prediction along z axis at time k*
$\hat{V}_{x\ k-1}$	*= Estimated velocity along x axis at time k-1 (EKF output)*
$\hat{V}_{y\ k-1}$	*= Estimated velocity along y axis at time k-1 (EKF output)*
$\hat{V}_{z\ k-1}$	*= Estimated velocity along z axis at time k-1 (EKF output)*
$\hat{\phi}_{k-1}$	*= Estimated roll at time k-1 (EKF output)*
$\hat{\theta}_{k-1}$	*= Estimated pitch at time k-1 (EKF output)*
$\hat{\psi}_{k-1}$	*= Estimated heading at time k-1 (EKF output)*
$\tilde{a}_{x\ k}$	*= x-axis acceleration + sampling noise*
$\tilde{a}_{y\ k}$	*= y-axis acceleration + sampling noise*
$\tilde{a}_{z\ k}$	*= z-axis acceleration + sampling noise*
g	*= Gravitational Constant 9.81 m/s^2*
T_s	*= Sampling Period*

The DCM, used to rotate the rover attitude Euler angles (ϕ, θ, ψ) such that the rover is aligned with the Cartesian axes is shown in Equation 7. The trigonometric functions *cosine* and *sine* are abbreviated as *c* and *s*, respectively.

$$DCM(\phi,\theta,\psi) = \begin{bmatrix} \mathrm{c}\,\psi\,\mathrm{c}\,\theta & -\mathrm{s}\,\psi\,\mathrm{c}\,\phi + \mathrm{c}\,\psi\,\mathrm{s}\,\theta\,\mathrm{s}\,\phi & \mathrm{s}\,\psi\,\mathrm{s}\,\phi + \mathrm{c}\,\psi\,\mathrm{s}\,\theta\,\mathrm{c}\,\phi \\ \mathrm{s}\,\psi\,\mathrm{c}\,\theta & \mathrm{c}\,\psi\,\mathrm{c}\,\phi + \mathrm{s}\,\psi\,\mathrm{s}\,\theta\,\mathrm{s}\,\phi & -\mathrm{c}\,\psi\,\mathrm{s}\,\phi + \mathrm{s}\,\psi\,\mathrm{s}\,\theta\,\mathrm{c}\,\phi \\ -\mathrm{s}\,\theta & \mathrm{c}\,\theta\,\mathrm{s}\,\phi & \mathrm{c}\,\theta\,\mathrm{c}\,\phi \end{bmatrix} \tag{7}$$

Where:

ϕ	*= Estimated roll (EKF output)*
θ	*= Estimated pitch (EKF output)*
ψ	*= Estimated heading (EKF output)*

Equation 8 [18] shows the integration of the EKF's previous estimate of velocity to predict the 3-axis position.

$$\begin{bmatrix} \hat{x}_k^- \\ \hat{y}_k^- \\ \hat{z}_k^- \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1} \\ \hat{y}_{k-1} \\ \hat{z}_{k-1} \end{bmatrix} + \begin{bmatrix} \hat{V}_{x\,k-1} \\ \hat{V}_{y\,k-1} \\ \hat{V}_{z\,k-1} \end{bmatrix} T_s \tag{8}$$

Where:

$\hat{x}_k^-$	*= Estimated position prediction along x axis at time k*
$\hat{y}_k^-$	*= Estimated position prediction along y axis at time k*
$\hat{z}_k^-$	*= Estimated position prediction along z axis at time k*
$\hat{x}_{k-1}$	*= Estimated position along x axis at time k-1 (EKF output)*
$\hat{y}_{k-1}$	*= Estimated position along y axis at time k-1 (EKF output)*
$\hat{z}_{k-1}$	*= Estimated position along z axis at time k-1 (EKF output)*
$\hat{V}_{x\,k-1}$	*= Estimated velocity along x axis at time k-1 (EKF output)*
$\hat{V}_{y\,k-1}$	*= Estimated velocity along y axis at time k-1 (EKF output)*
$\hat{V}_{z\,k-1}$	*= Estimated velocity along z axis at time k-1 (EKF output)*
T_s	*= Sampling Period*

4.3.2. Rover State Measurement

The measurement step calibrates the estimates against a bounded-error input. The three-state attitude estimate is calibrated using the IMU's accelerometers and tilt-corrected heading that is determined using the IMU’s magnetometers corrected with roll and pitch estimates from the prior run of the EKF. The position estimate is calibrated with three-axis position input from a GPS. The velocity estimate is calibrated with the three-axis velocity reported by the GPS. Each of these inputs exhibits Gaussian noise in addition to the sensor output. This process provides a better estimate by combining the *predicted* states, which provide a better short-term estimate with the *measured* states that provide a better long-term estimate and have bounded error characteristics.

4.3.2.1. GPS Coordinate Measurement

The GPS measurement is taken from the NovAtel OEM615 GPS. The GPS provides the rover coordinates in ECEF format. This format was chosen because it provides a unique point on the globe that can be easily converted to East-North-Down (END) local-plane coordinates to correlate to inertial information from the IMU. This allows the predicted position and velocity along the rover axes (described in Section 4.3.1) to be transformed into movements in the local Cartesian plane. The END coordinate system provides the correlation between GPS and inertial state estimation, which is necessary for the EKF. The remainder of this section describes the relationships between these two coordinate systems, as well as the Geodetic system used to report the rover state to the OCU and calculate range and heading between the current position and goal positions provided for telemetry and input to the proposed autonomous navigation described later in Section 6.2.4.

The fusion of inertial and GPS information, and the useful reporting of the output requires coordinate transformations. This work uses three different coordinate systems:

- Geodetic: Used to report position to operators in easily-understood format
- ECEF: Commonly used by GPS systems due to independence from ellipsoid models. ECEF forms the ‘bridge’ between Geodetic coordinates and END coordinates
- END: A flat plane tangent to the Earth surface that allows for geometric integrations of acceleration to derive velocity and position

Geodetic coordinates are common for human interfaces. Geodetic coordinates are reported in terms of latitude, longitude, and altitude. The latitude ranges from -90° to 90° and measures the rotational angle between the Prime Meridian plane and the normal of the elliptical plane that passes through the coordinate point. The longitude ranges from -180° to 180° and measures the rotational angle between the equatorial plane and the normal of the elliptical plane that passes through the coordinate point. The altitude is the local vertical distance between the ellipsoid model of the earth and the coordinate point [38]. While geodetic coordinates are most common in communicating position, they do not describe a Cartesian plane which is necessary to translate movements of the rover to position change.

The ECEF coordinate system rotates with the Earth, around its spin axis. This means every point on the Earth has a fixed set of coordinates, described by a three-element vector containing offsets from the *x*, *y*, and *z* axes [38]. This model follows the elliptical surface of the Earth and therefore does not lend itself well to Cartesian calculations rover position change. However, because it describes a position on the Earth's surface, and not the angle of a vector from planes intersecting the Earth, it is used as an intermediary coordinate system between Geodetic coordinates and the Cartesian plane described by END coordinates.

END coordinates describe a plane, fixed to the Earth's surface. This plane is aligned such that the *x* axis points north, the *y* axis is orthogonal and points east, and the *z* axis is perpendicular to the plane and points to the Earth's center [38]. The END coordinate system provides a relationship between a north/east oriented plane and the rover body axes that lets us define three-element vectors describing rover position, velocity, and acceleration with respect to the *x*, *y*, and *z* axes. This allows for simple Cartesian calculations to determine these states.

Figure 4-7 shows the relationship between the END plane (shown in green), the ECEF axes (shown in blue), and Geodetic Latitude, longitude and altitude (shown in orange). Figure 4-8 shows the rover body axes and their relationship to the END plane (shown in green).

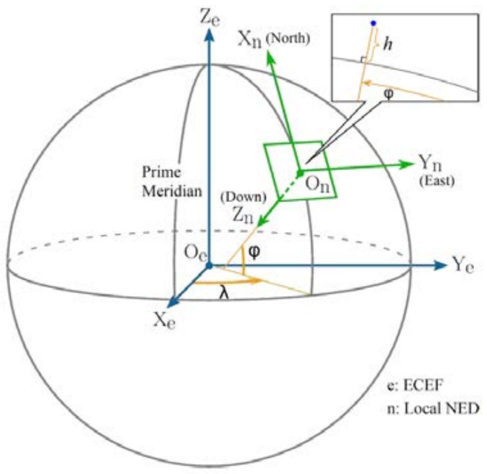


Figure 4-7: Relationship between Geodetic, ECEF and END Coordinates [38]

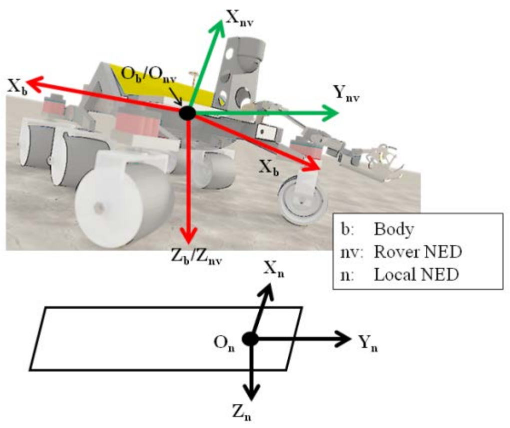


Figure 4-8: Relationship between Local NED, rover NED, and Body Coordinates

4.3.2.2. Attitude Measurement Derivation

Accelerometers sense acceleration due to rover movement as well as constant gravitational acceleration. While the two sources of acceleration cannot be differentiated by the sensor, the attitude can be approximated using the accelerometers' measurement of constant gravity. This approximation is based on the assumption that the rover is normally operating at a constant velocity. This assumption holds for most operations of the rover, and small accelerations experienced when the rover starts and stops movement are attenuated by the EKF [16]. Equation 9 shows the calculation of the pitch of the rover. The calculated *pitch* of the rover is used to determine the *roll* of the rover in Equation 10. These outputs are used to calibrate the *roll* and *pitch* predicted by the gyroscopes. Yaw or *heading* cannot be determined using the accelerometer and thus is determined using the magnetometers (detailed in Section 4.3.2.3)

$$\tilde{\theta}_k = sin^{-1}\left(\frac{\tilde{a}_{x\,k}}{g}\right) \tag{9}$$

Where:

$\tilde{\theta}_k$	= *Approximated Pitch at time k + sampling noise*
$\tilde{a}_{x\,k}$	= *x-axis acceleration at time k + sampling noise*
g	= *Gravitational Acceleration Constant 9.8065 m/s²*

$$\tilde{\phi}_k = \sin^{-1}\left(\frac{-\tilde{a}_{y\,k}}{g\cos\tilde{\theta}_k}\right) \tag{10}$$

Where:

$\tilde{\phi}_k$	*= Approximated Roll at time k + sampling noise*
$\tilde{a}_{y\,k}$	*= y-axis acceleration at time k + sampling noise*
$\tilde{\theta}_k$	*= Approximated Pitch at time k + sampling noise*
g	*= Gravitational Acceleration Constant 9.8065 m/s^2*

4.3.2.3. Compass Measurement Derivation

The heading is calculated in the horizontal plane; however the rover often is not level. In fact, when the rover is on level ground, it is pitched forward. This requires the magnetometer readings of the Earth's magnetic field, taken on-board the rover, to be rotated back to level. This process is called *tilt-compensation*. Equation 11 rotates the magnetometer readings in the *x*, *y*, and *z* rover body axes (as shown in Figure 4-8) to determine the *x* axis magnetic field in the horizontal plane [39]. The tilt compensation calculations take advantage of the increased accuracy provided by the EKF by utilizing the EKF output of roll and pitch [39].

$$\tilde{X}_{h\,k} = \tilde{m}_{x\,k}\cos\hat{\phi}_{k-1} + \tilde{m}_{y\,k}\sin\hat{\phi}_{k-1}\cos\hat{\theta}_{k-1} - \tilde{m}_{z\,k}\cos\hat{\phi}_{k-1}\sin\hat{\theta}_{k-1} \tag{11}$$

Where:

$\tilde{X}_{h\,k}$	*= Tilt corrected horizontal plane x axis magnetic field reading + sampling noise*
$\tilde{m}_{x\,k}$	*= Rover x-axis magnetic field + sampling noise*
$\tilde{m}_{y\,k}$	*= Rover y-axis magnetic field + sampling noise*
$m_{z\,k}$	*= Rover z-axis magnetic field + sampling noise*
$\hat{\phi}_{k-1}$	*= Estimated roll at time k-1 (EKF output)*
$\hat{\theta}_{k-1}$	*= Estimated pitch at time k-1 (EKF output)*

Equation 12 rotates the magnetometer readings in the *x*, *y*, and *z* rover body axes (as shown in Figure 4-8) to determine the *y* axis magnetic field in the horizontal plane [39].

$$\tilde{Y}_{h\,k} = \tilde{m}_{y\,k} \cos \hat{\phi}_{k-1} + \tilde{m}_{z\,k} \sin \hat{\phi}_{k-1} \tag{12}$$

Where:

$\tilde{Y}_{h\,k}$	*= Tilt corrected horizontal plane y axis magnetic field reading + sampling noise*
$\tilde{m}_{y\,k}$	*= Rover y-axis magnetic field + sampling noise*
$m_{z\,k}$	*= Rover z-axis magnetic field + sampling noise*
$\hat{\phi}_{k-1}$	*= Estimated roll at time k-1 (EKF output)*
$\hat{\theta}_{k-1}$	*= Estimated pitch at time k-1 (EKF output)*

Electronic magnetometers are sensitive to the presence of both *hard-iron* and *soft-iron* effects, and must be calibrated to remove these effects in order to provide an accurate heading. During development the *soft-iron* effects were minimal; s*oft-iron* effects are caused by materials like iron and nickel being near the sensor. This rover uses very few ferrous materials and they appear to have an insignificant effect on the sensor readings. Only significant *hard-iron* effects were seen on the magnetometer readings, therefore its correction is discussed here. *Hard-iron* effects on magnetometer readings are caused by magnetic materials near the magnetometer, such as the magnets in the rover motors. These materials create a constant additive value to the Earth's magnetic field [40]. The offsets created by the *hard-iron* effect are determined by slowly turning the rover in place and collecting tilt-compensated $\tilde{X}_h$ and $\tilde{Y}_h$ values. The maximum and minimum values of both $\tilde{X}_h$ and $\tilde{Y}_h$ are determined and used to determine the offsets using Equations 13 and 14 below.

$$\alpha = \frac{\left(\tilde{\boldsymbol{X}}_{\boldsymbol{h}\,max} + \tilde{\boldsymbol{X}}_{\boldsymbol{h}\,min}\right)}{2} \tag{13}$$

Where:

α	*= x axis offset*
$\tilde{X}_h$	*= Vector Tilt corrected horizontal plane x axis magnetic field readings + sampling noise (taken over full-circle rotation)*

$$\beta = \frac{(\tilde{\boldsymbol{Y}}_{\boldsymbol{h}\,max} + \tilde{\boldsymbol{Y}}_{\boldsymbol{h}\,min})}{2} \tag{14}$$

Where:

β	= *y axis offset*
$\tilde{\boldsymbol{Y}}_{\boldsymbol{h}}$	= *Vector Tilt corrected horizontal plane y axis magnetic field readings + sampling noise (taken over full-circle rotation)*

Once these additive offsets have been determined through calibration, they are subtracted from the tilt-corrected readings during the heading calculation, as shown in Equations 15 and 16.

Figure 4-9 illustrates the *hard-iron* calibration process. The blue dots show the magnetic field data prior to calibration, the red dots show the magnetic field data, corrected for the *hard-iron* offset and centered at 0.

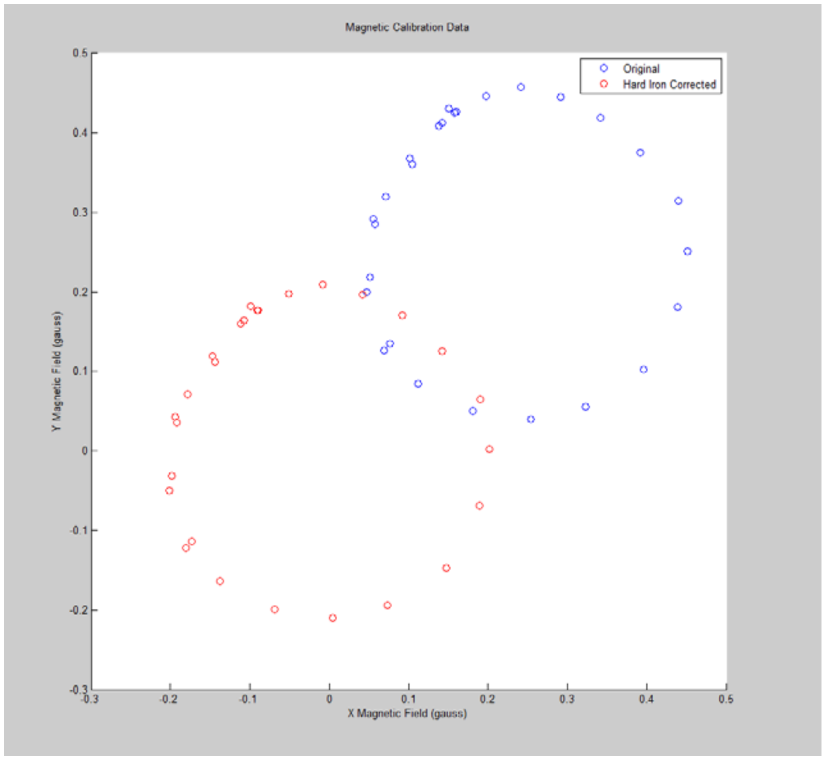


Figure 4-9: Magnetometer Calibration Example

The tilt and *hard-iron* compensated *x* and *y* axis magnetic fields are used to determine the magnetic heading of the rover. Simply put, the heading can be calculated by Equation 15 [39].

$$\tilde{\psi}_k = \tan^{-1}\left(\frac{\tilde{Y}_{h\,k} - \beta}{\tilde{X}_{h\,k} - \alpha}\right) \tag{15}$$

Where:

$\tilde{\psi}_k$	*= Magnetic heading at time k + sampling noise*
$\tilde{Y}_{h\,k}$	*= Tilt corrected horizontal plane y axis magnetic field reading + sampling noise*
$\tilde{X}_{h\,k}$	*= Tilt corrected horizontal plane x axis magnetic field reading + sampling noise*
α	*= x axis offset*
β	*= y axis offset*

However, Equation 13 does not account for the tangent function not being valid at angles greater than 180°, nor does it allow for $\tilde{X}_{h\,k} - \alpha$ to equal 0 (division by zero error). However the Microsoft .NET4.5 framework contains a function called *atan2* that provides the arctangent defined in the range of –π to π [41]. Equation 16 describes the *atan2* output for points on the boundaries and each quadrant, described in the input variables specific to this work corrected to place the compass heading in the 0 - 2π range for calculation purposes, which converts to the 0° - 360° range for reporting purposes .

$$\begin{gathered}
if\left(\tilde{X}_{h\,k} - \alpha = 0, \tilde{Y}_{h\,k} - \beta > 0\right)\ then\ \ \tilde{\psi}_k = \frac{\pi}{2} \\
else\ if\left(\tilde{X}_{h\,k} - \alpha = 0, \tilde{Y}_{h\,k} - \beta < 0\right)\ then\ \ \tilde{\psi}_k = \frac{3\pi}{2} \\
else\ if\left(\tilde{X}_{h\,k} - \alpha < 0, \tilde{Y}_{h\,k} - \beta = 0\right)\ then\ \ \tilde{\psi}_k = \pi \\
else\ if\left(\tilde{X}_{h\,k} - \alpha \geq 0, \tilde{Y}_{h\,k} - \beta = 0\right)\ then\ \ \tilde{\psi}_k = 0 \\
else\ if\left(\tilde{X}_{h\,k} - \alpha > 0, \tilde{Y}_{h\,k} - \beta > 0\right)\ then\ \ \tilde{\psi}_k = \tan^{-1}\left(\frac{\tilde{Y}_{h\,k} - \beta}{\tilde{X}_{h\,k} - \alpha}\right) \\
else\ if\left(\tilde{Y}_{h\,k} - \beta < 0\right)\ then\ \ \tilde{\psi}_k = \tan^{-1}\left(\frac{\tilde{Y}_{h\,k} - \beta}{\tilde{X}_{h\,k} - \alpha}\right) + \pi
\end{gathered} \tag{16}$$

$$else\ if\left(\tilde{X}_{h\,k} - \alpha < 0, \tilde{Y}_{h\,k} - \beta > 0\right)\ then\ \ \tilde{\psi}_k = \frac{3\pi}{2} - \tan^{-1}\left(\frac{\tilde{Y}_{h\,k} - \beta}{\tilde{X}_{h\,k} - \alpha}\right)$$

Where:

$\tilde{\psi}_k$	*= Magnetic heading at time k + sampling noise*
$\tilde{Y}_{h\,k}$	*= Tilt corrected horizontal plane y axis magnetic field reading + sampling noise*
$\tilde{X}_{h\,k}$	*= Tilt corrected horizontal plane x axis magnetic field reading + sampling noise*
α	*= x axis offset*
β	*= y axis offset*

The heading calculation requires one more correction, once calculated. Depending on the rover's position on Earth it is subject to a fixed magnetic declination. Magnetic declination refers to the angle between magnetic and true north. This constant value was determined by referring to the *NOAA Declination Calculator* and subtracting the declination value from the heading calculated in Equation 16. The declination for the Morgantown, WV area is 9.09°W [42].

4.3.3. EKF Estimate Output

The output of the EKF is determined in Step 5 of the model detailed in Section 2.3.2. An excerpt from the EKF block diagram, highlighting this important step, is shown in Figure 4-10. This step describes the relationship of predicted state and measured state to determine the estimate.

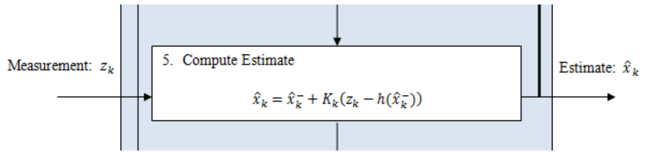


Figure 4-10: Extended Kalman Filter Estimate Output Diagram [16]

A key component is to determine the *residual*, or error, between the predicted and measured states. This is achieved by Equation 17, a crucial step to computing the estimate in Step 5 of the EKF shown in Figure 2-7 by subtracting the state-observation function vector from the measured state vector. In this case, the state observation function, $h(x)$, is simply a 9x9 identity matrix. $\boldsymbol{z_k}$ is a vector containing the nine-state measurement and $\boldsymbol{\hat{x}_k}$ is a vector containing the nine-state prediction.

Note the tilde above the six GPS inputs and three attitude measurements; this denotes the addition of a random measurement noise term. The dash superscript denotes a prediction, and the hat accent denotes an estimate.

$$residual = \boldsymbol{z}_k - h(\boldsymbol{\hat{x}}_k^-)$$

$$\boldsymbol{\hat{x}_k} = \left[\hat{x}_k^- \quad \hat{y}_k^- \quad \hat{z}_k^- \quad \hat{V}_{x\,k}^- \quad \hat{V}_{y\,k}^- \quad \hat{V}_{z\,k}^- \quad \hat{\phi}_k^- \quad \hat{\theta}_k^- \quad \hat{\psi}_k^-\right]^{\boldsymbol{T}} \tag{17}$$

$$\boldsymbol{z_k} = \left[\tilde{x}_k \quad \tilde{y}_k \quad \tilde{z}_k \quad \tilde{V}_{x_k} \quad \tilde{V}_{y_k} \quad \tilde{V}_{z_k} \quad \tilde{\phi}_k \quad \tilde{\theta}_k \quad \tilde{\psi}_k\right]^{\boldsymbol{T}}$$

Where:

$\hat{x}_k^-$	*= Estimated position prediction along x axis at time k*
$\hat{y}_k^-$	*= Estimated position prediction along y axis at time k*
$\hat{z}_k^-$	*= Estimated position prediction along z axis at time k*
$\hat{V}_{x\,k}^-$	*= Estimated velocity prediction along x axis at time k*
$\hat{V}_{y\,k}^-$	*= Estimated velocity prediction along y axis at time k*
$\hat{V}_{z\,k}^-$	*= Estimated velocity prediction along z axis at time k*
$\hat{\phi}_k^-$	*= Estimated roll prediction at time k*
$\hat{\theta}_k^-$	*= Estimated Pitch prediction at time k*
$\hat{\psi}_k^-$	*= Estimated Heading prediction at time k*
$\tilde{x}_k$	*= Measured GPS x position + noise*
$\tilde{y}_k$	*= Measured GPS y position + noise*
$\tilde{z}_k$	*= Measured GPS z position + noise*
$\tilde{V}_{x_k}$	*= Measured GPS x velocity + noise*
$\tilde{V}_{y_k}$	*= Measured GPS y velocity + noise*
$\tilde{V}_{z_k}$	*= Measured GPS z velocity + noise*
$\tilde{\phi}_k$	*= Calculated accelerometer-based roll + noise*
$\tilde{\theta}_k$	*= Calculated accelerometer-based pitch + noise*
$\tilde{\psi}_k$	*= Calculated tilt-corrected magnetometer-based heading + noise*

The *residual* term is part of Equation 18, which computes the estimated state for input to the next iteration of the EKF and output. Essentially the current state is the predicted state plus a correction factor determined by multiplying the *residual* by the Kalman gain, K_k.

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_k^-)) \quad (18)$$

4.3.4. Kalman Gain and EKF Tuning

The Kalman gain is the lynchpin component of the Kalman filter, and determines the ratio of predicted state versus measured state in the estimate. The Kalman gain is determined by calculating the error covariance for each iteration of the EKF and adding two fixed *tuning* term vectors, $\boldsymbol{Q}$ and $\boldsymbol{R}$. The $\boldsymbol{Q}$ and $\boldsymbol{R}$ vectors correspond to the expected covariance of Gaussian noise, or amount of uncertainty, present in the prediction and measurement, respectively. These factors were found through experimentation with the hardware. Long term data-collections of each prediction and measurement were performed and the variance of each set of data were recorded. These were used as the starting point, and through experimentation, tailored to produce the desired results.

4.3.5. NAS Output

The NAS estimated states are output to the RCU software at a rate of 10Hz. The reported NAS states, that are sent to the OCU as telemetry, include Geodetic Latitude and Longitude (derived from the END coordinates maintained internal to the EKF), ECEF 3-axis velocity (used for performance testing of the EKF), and roll, pitch, and heading.

Chapter 5
NAS Testing and Results

The NAS must provide accurate state data in order to support autonomous navigation and useful telemetry to the operator. The NAS test was designed to ensure that the attitude and velocity states provided useful data to the position estimation. The test was also designed to verify the heading state, as the heading and position are the primary data necessary for autonomous navigation and position telemetry. A series of tests were carried out to evaluate the effectiveness of the EKF to fuse data from gyroscopes, accelerometers, magnetometers, and a GPS receiver to provide the 9-state estimate. The attitude and velocity tests were a sanity check and simply provide feedback to the effectiveness of the EKF in fusing the sensor data. The heading and position test compared data from the NAS to data collected form a Hemisphere V100 high-precision GPS receiver, shown in Figure 5-1 [43].

Figure 5-1: Hemisphere V100 GPS receiver [43]

The V100 incorporates two GPS antennas, positioned at a known aperture, that are used in Differential Global Position System (DGPS) reception in conjunction with inertial aiding [43]. Utilizing DGPS, the V100 is capable of providing GPS position with an accuracy of 0.6m with a 95% confidence interval [43]. The V100 is also capable of providing compass heading with an accuracy of 0.3° [43].

These outputs were used as comparative data for the NAS output. The V100 is only capable of 1Hz output, therefore comparison between it and the NAS required the NAS data to be downsampled to match.

It was observed early in the testing that interference from the other electronics on-board the rover interfered with both GPS receivers, preventing them from being able to get a position lock. Subsequent testing of the two systems in a collocated configuration showed further interference between the Hemisphere V100 GPS and the NAS. This did not always prevent position lock but severely degraded the accuracy of the position feedback. Efforts to mitigate the electronic interference included mounting the NAS board inside of an enclosed aluminum box, shown in Figure 5-2. This creates a Faraday cage, blocking external interference from the board, without blocking magnetic field data. A non-metal rover electronics cover was replaced with an aluminum cover, fastened to the aluminum electronics case with steel fasteners to create a second Faraday cage. These efforts had a limited benefit, allowing the GPS to get lock, but still with degraded position. Upon these observations, the decision was made to test the two systems on the same test course, but not simultaneously. Rather, one system at a time was tested on the course, and then the other system was tested on the same course. This provided comparative data without the negative effects of the electronic interference.

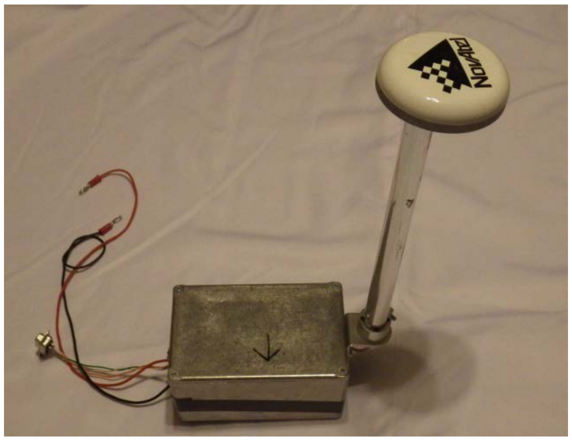


Figure 5-2: NAS enclosed in Aluminum Box

Tests were designed to specifically test each state output by the NAS. The pitch and roll attitude states test and results are described in Section 5.1. The testing of the heading attitude state and results is presented in Section 5.1.1.1. The velocity state testing and results is shown in Section 5.1.1.2.The position testing and results are shown in Section 5.1.1.3. In order to demonstrate the benefit of using inertial data during degraded GPS reception to predict the rover position, a test featuring a GPS signal loss is presented in Section 5.1.1.4.

5.1. Attitude Testing and Results

The roll and pitch were evaluated to determine the noise rejection, on the gyroscopes and accelerometers, provided by the EKF. In order to establish truth without additional attitude measuring equipment (attitude was not the main focus and at time of work no additional attitude sensing hardware was available), two tests were carried out. First the NAS was set level, according to a bubble level, then sensor and EKF data were collected for five minutes. The gyroscope component of the attitude estimation was recorded two ways to show the effect of the EKF and how the data is processed. First the instantaneous discretized gyroscope input is recorded. In addition, a gyroscope-only estimate of attitude, determined by integrating the gyroscope inputs over the test time, is shown to demonstrate possible integration error accumulating over time. This test shows the impact of the accelerometers and the noise from the gyroscopes. Figure 5-3 shows the plot of the pitch estimate for the integrated gyroscope data, , the accelerometer-only attitude, and the fusion of the gyroscope and accelerometer data using an EKF. These plots demonstrate the accumulating integration error, created by integrating noisy inputs over time, shown by the integrated gyroscope estimate of attitude. The difficulty in determining the difference between discrete gyroscope input, accelerometer-only estimate of attitude, and EKF estimate of attitude further illustrates how even small sensor data errors can accumulate to become large errors as well as the effectiveness of the EKF in rejecting this sensor noise. In order to better illustrate the effectiveness of the EKF, Figure 5-4 shows a close up of the first 4 seconds of data shown in Figure 5-3. This plot better illustrates the effect of the EKF in utilizing the gyroscope predicted pitch to reduce bias noise from the accelerometers.

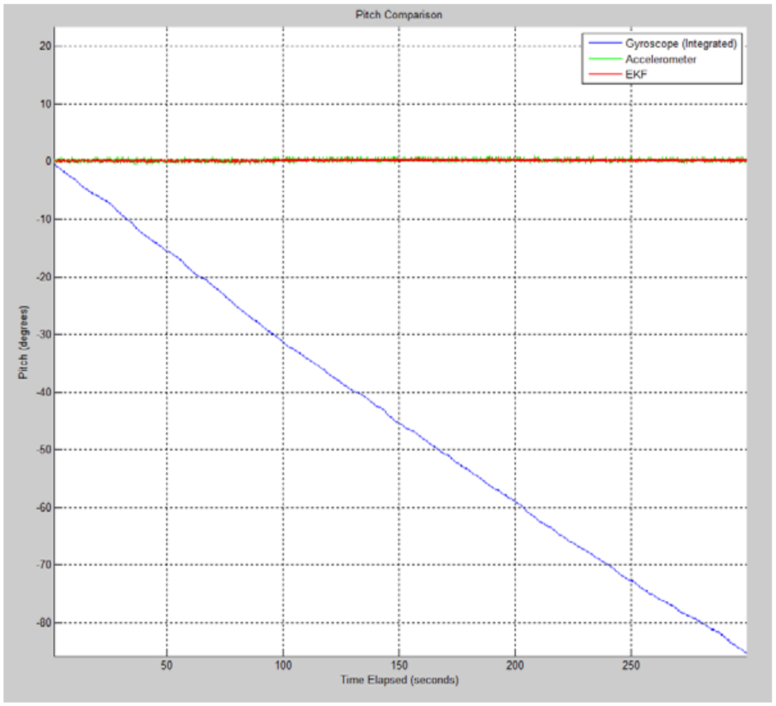


Figure 5-3: Pitch Estimation Comparison of Sensors vs. EKF for Static Level Position

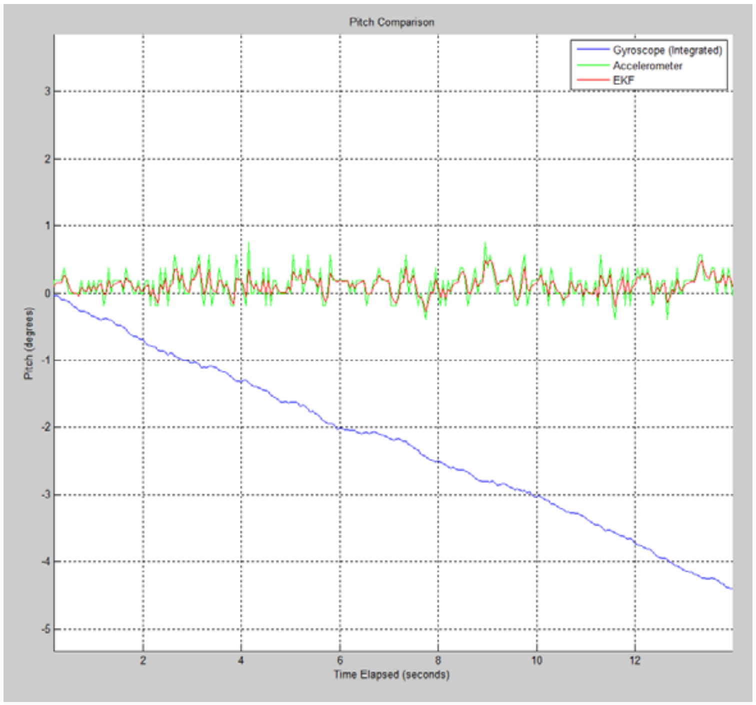


Figure 5-4: Close-up of Pitch Comparison of Sensors vs. EKF for Static Level Position

The second test was intended to demonstrate the increase in responsiveness in gyroscope attitude data versus accelerometer attitude data, and how the EKF can be more responsiveness to changes in attitude, while rejecting integration error, or *drift*, in the gyroscope data. The test was performed by oscillating the NAS around the *y*-axis from approximately ±30°, and collecting sensor and EKF data. This test was performed by hand and is not intended to oscillate perfectly. However, by plotting the integrated gyroscope data, discretized gyroscope data, the accelerometer-only attitude, and the fusion of the gyroscope and accelerometer data using an EKF in the same graph, the EKF can be evaluated for noise rejection and responsiveness. This plot is shown in Figure 5-5 with a close up on one period of oscillation shown in Figure 5-6.

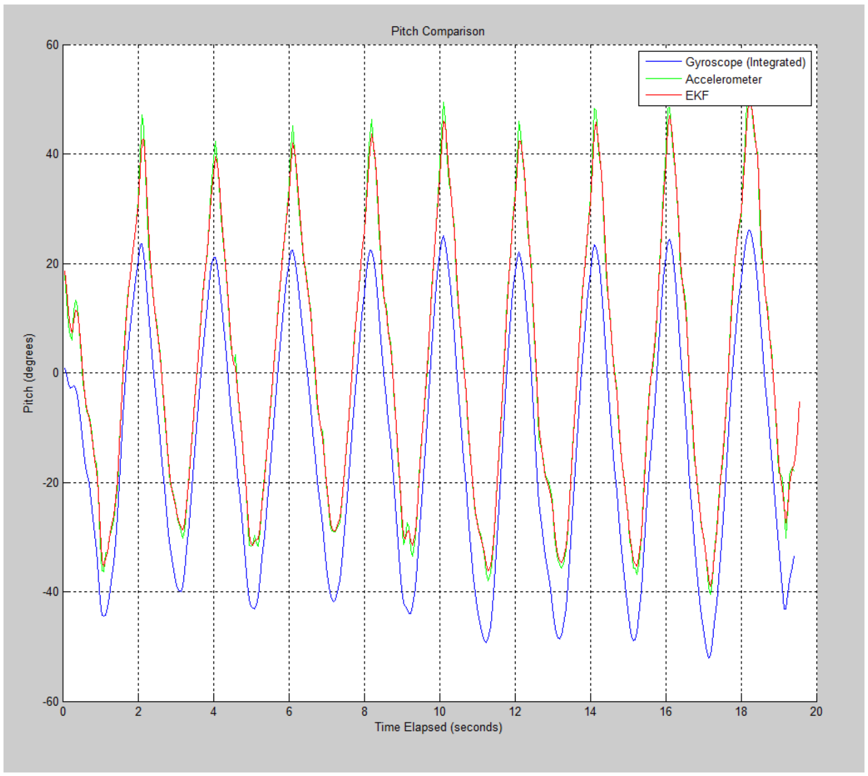


Figure 5-5: Pitch Estimation Comparison of Sensors vs. EKF for Oscillating Position

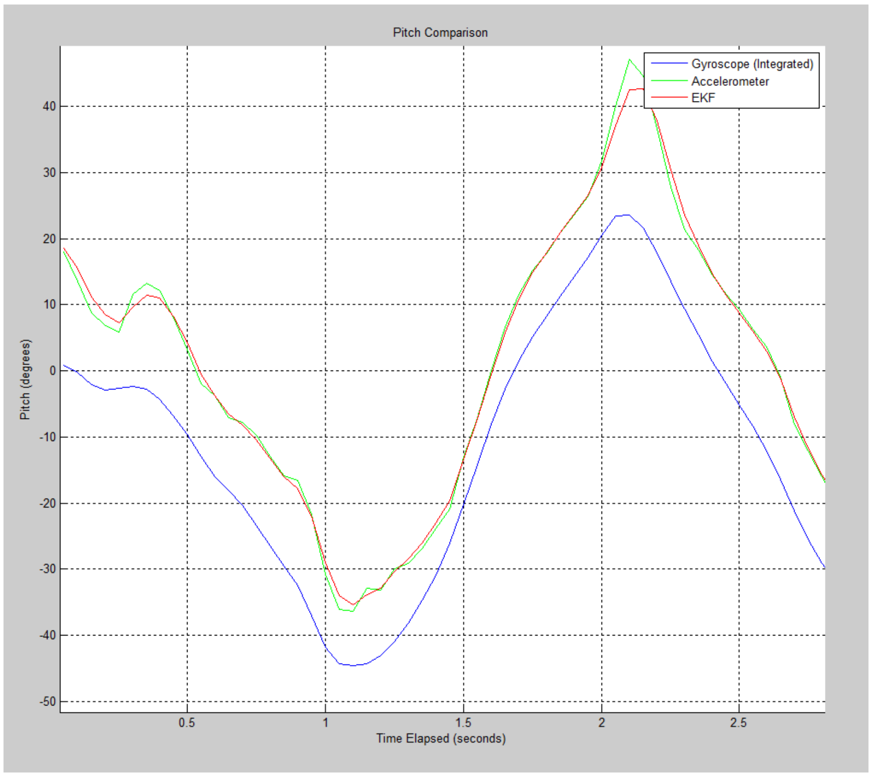


Figure 5-6: Comparison of Single Period of Oscillating Pitch Estimation of Sensors vs. EKF

This data analysis is identical to the static attitude test, and demonstrated some additional phenomena. First the major integration error in the gyroscope-only estimation of attitude is absent, due to the motion being much greater than the noise, and essentially washing it out. Second, the gyroscope-only estimate is offset from the accelerometer and EKF data by approximately 18°. This is due to the gyroscope having no *a priori* knowledge of the attitude and therefore starting to integrate at 0. The EKF estimate of attitude very-closely follows that of the accelerometer only data, with the exception of the transition from positive pitch to negative pitch and vice-versa. Here it is clear that the EKF more closely follows the trend of the gyroscope data, more accurately estimating this transition. Gu's work with unmanned aerial vehicles (UAV), only used the gyroscope-estimated attitude to save computer processing resources. This compromise works well in a highly dynamic system, such as a UAV, as demonstrated in Figure 5-5. However, for ground vehicles, such as our rover, which move slowly and may sit still for extended periods of time the EKF rejects the integration error that can accumulate by integrating gyroscope-data to provide attitude estimation.

5.1.1.1. Heading Testing and Results

The heading is a crucial component of an autonomous navigation system. The heading tests concentrated on comparing heading output of the NAS versus the heading output of the Hemisphere V100 GPS during motion tests. During the test each system was walked around a pre-determined course and data were logged. These data included heading information from both the V100 and the NAS. The averaged data from five test trials of the NAS are shown in Figure 5-7. The plot compares averaged NAS (EKF) data, averaged magnetic heading, and averaged V100 data.

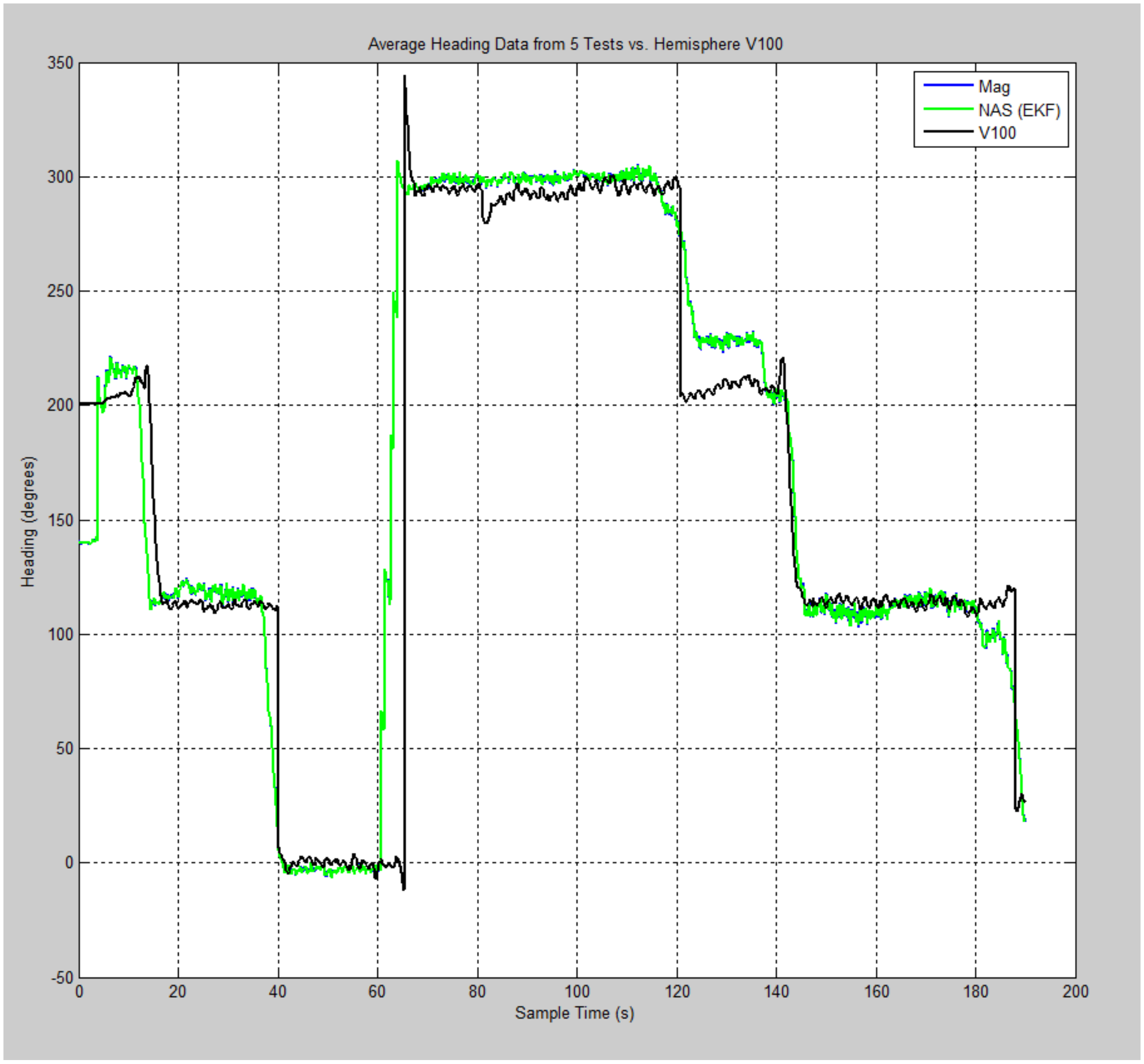


Figure 5-7: Heading Estimation Comparison between NAS and Hemisphere V100

The averaged data is slightly more noisy that the V100. The V100, when it has differential GPS lock, offers stated heading accuracy of 0.3° [43] and for each of the orientations it varies by approximately ±1°. The NAS does not require GPS to ascertain heading but its output is noisier than that of the V100. For each orientation, the NAS heading varies by approximately ±2.5°.

Static data, taken from corner one of the test course, was plotted to determine the effect of motion on heading estimation. From the plot, shown in Figure 5-7, it is clear that motion does introduce some noise. The approximate variation in NAS heading is ±1°, which still falls short of the V100's 0.3° accuracy (more clearly illustrated in Figure 5-8). The average heading difference between the NAS and the V100 is approximately 2.3° and can be attributed to a difference in initial orientation; both were aimed by eyesight at corner two.

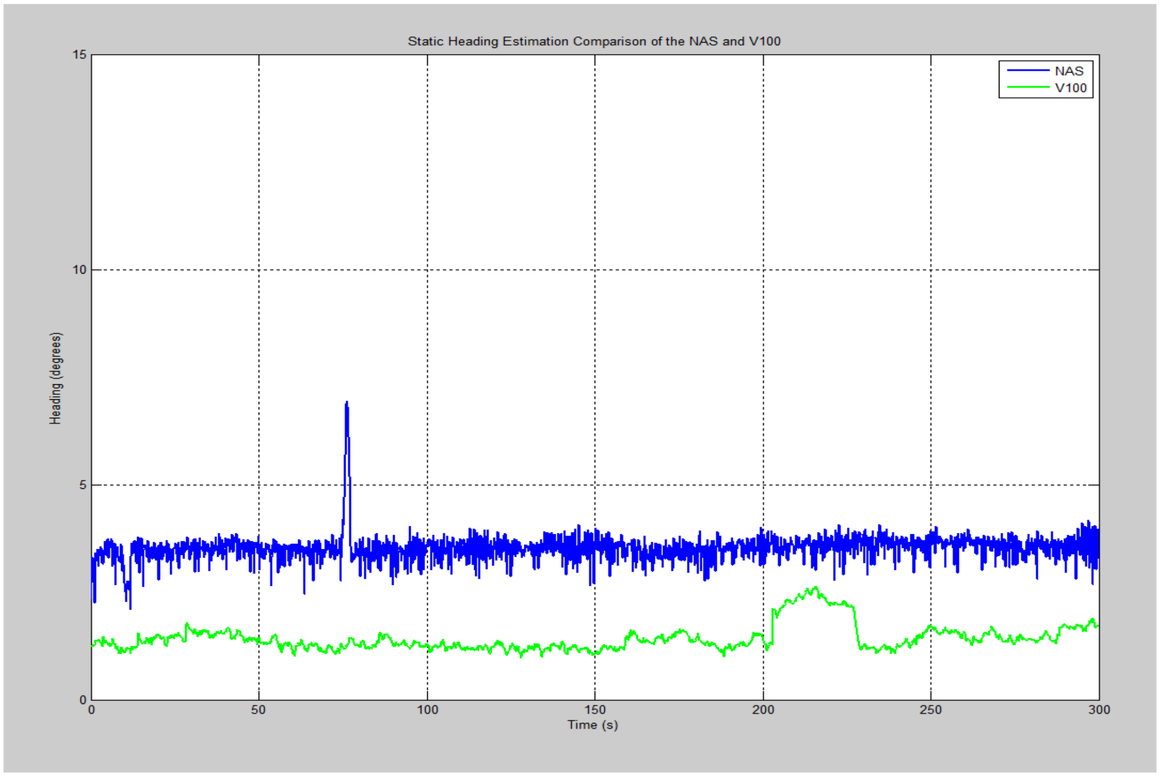


Figure 5-8: Heading Estimation Comparison of NAS and Hemisphere V100 data

5.1.1.2. Velocity Testing and Results

Velocity testing centered on comparing EKF estimated velocity versus GPS only velocity. The EKF velocity is used to predict the rover position in the EKF. Figure 5-9 shows the comparison plot between the magnitude of velocity in the END plane, determined by finding the magnitude of the XY vector for the GPS-reported velocity and the EKF estimated velocity. The plot shows a small noise reduction provided by the EKF.

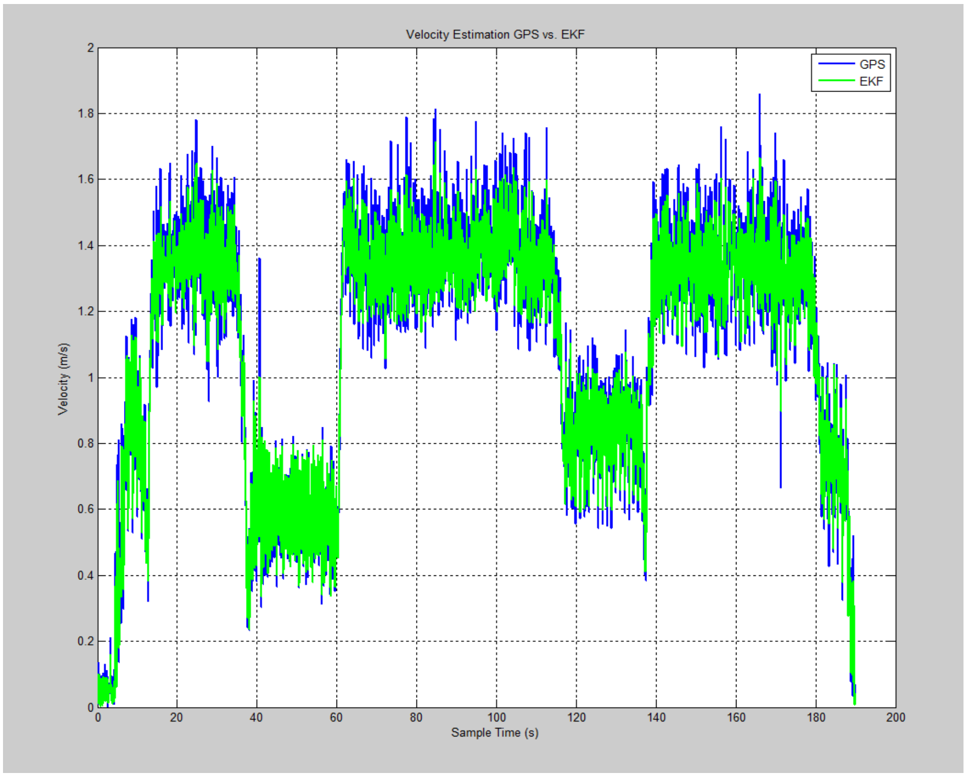


Figure 5-9: Velocity Estimation Comparison between EKF and GPS-only

5.1.1.3. Position Testing and Results

The NAS uses the estimated attitude and velocity states in conjunction with the GPS to estimate the position. In addition to individually looking at all of the inputs to the position estimate, extensive testing was performed on the NAS position output. This was performed through a sequence of tests, carried out in the WVU Coliseum parking lot. The test area, shown in Figure 5-10, is approximately 30m (illustrated by the gold line) by 100m (illustrated by the blue line). This size was chosen to mimic a large portion of the NASA Johnson Space Center (JSC) Rock Yard, shown in Figure 5-11.

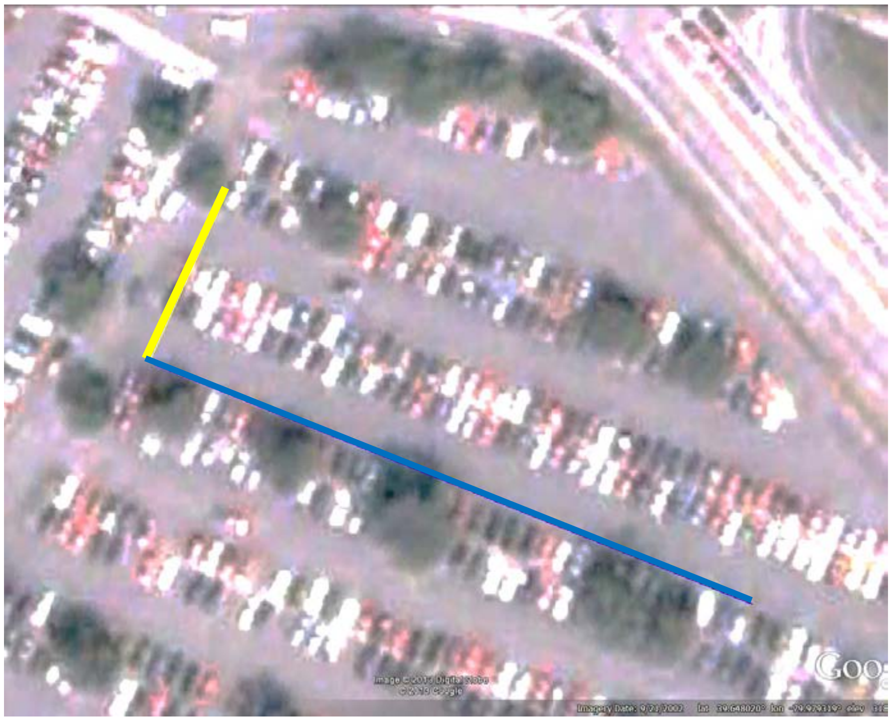


Figure 5-10: WVU Coliseum Parking Lot Test Area

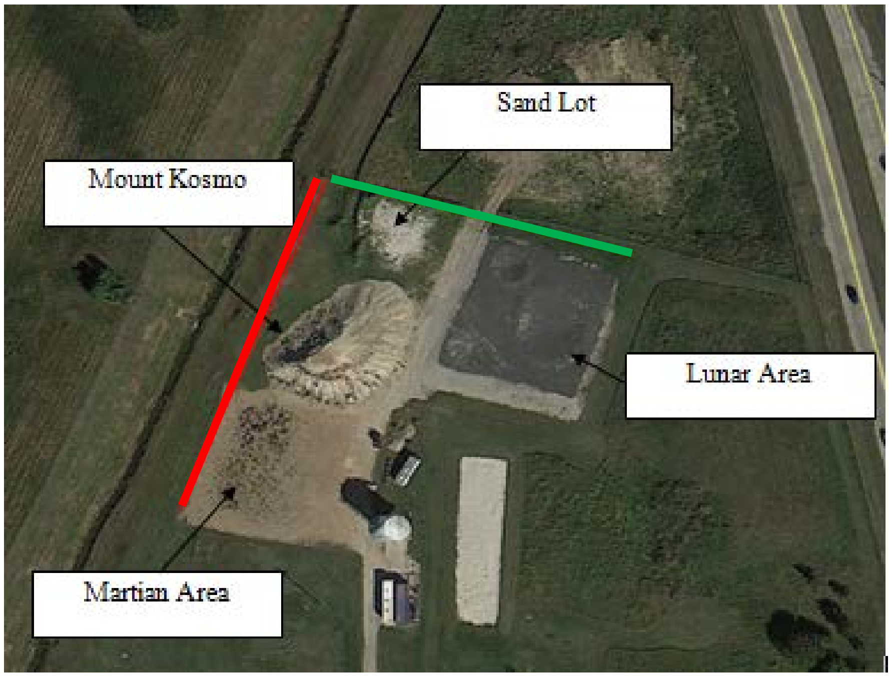


Figure 5-11: NASA JSC Rock Yard

The JSC Rock Yard covers an approximate 100 x 100 meter area, illustrated by the red and green lines in Figure 5-11. The Rock Yard comprises four distinctly different surfaces or area of interest. These include:

- Mount Kosmo: a 3 meter tall artificial mountain
- Martian Area: A rock Strew surface that is similar to that of Mars
- Sand Lot: A loose sandy surface
- Lunar Area: A crater filled gravel area imitating the Lunar surface

These areas vary in size from approximately 30m x 50m to 50m x 50m and are adjacent to each other. The test area chosen for the rover position simulates the surface area of two adjacent areas area to reflect the area needed to explore one area and navigate to the next.

The first test evaluates the GPS error between the Hemisphere V100 and the NAS. Four points' coordinates were surveyed using the Hemisphere V100; these four points mark four corners of the test path that the systems traverse in all subsequent tests. The surveyed data from the V100 was averaged to form the center coordinates of a 2m radius circle; this was plotted to determine if the NAS data fell within a 2m threshold. In observations, the V100 data varied by less than .12 meters from this average. This is illustrated in Figure 5-12. Five minutes of data for each position was collected with the NAS and plotted on the figure. All of the points fell within the 2m error threshold, with the maximum average error being approximately 1.4m at Corner 1.

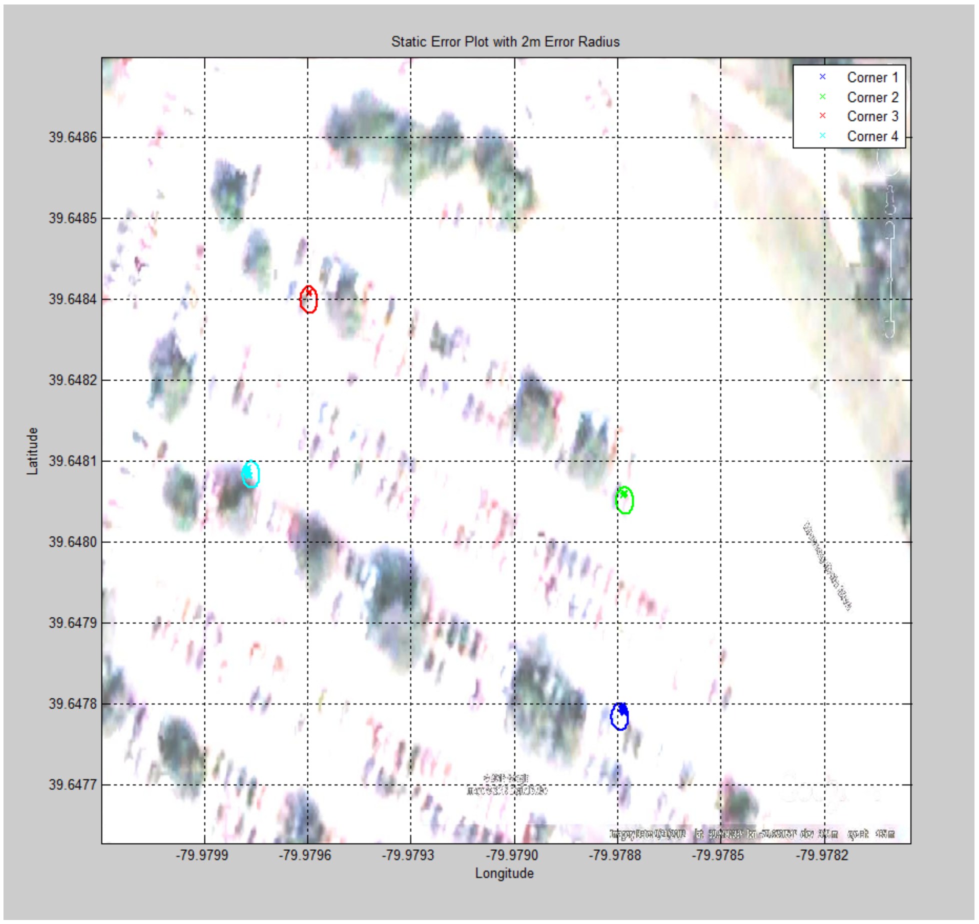


Figure 5-12: Comparison of Static NAS data to Hemisphere V100 Surveyed Locations

The second test involved traversing the test path, starting and finishing at the same point inside of the path, with both systems. Each system was carried around the test path five times to ensure that the positions were consistent. The surveyed corner data from the Hemisphere V100 was used to determine the truth data for the course. Figure 5-13 shows the data collected on each of the five traversals by the NAS, plotted against the truth data taken with the V100.

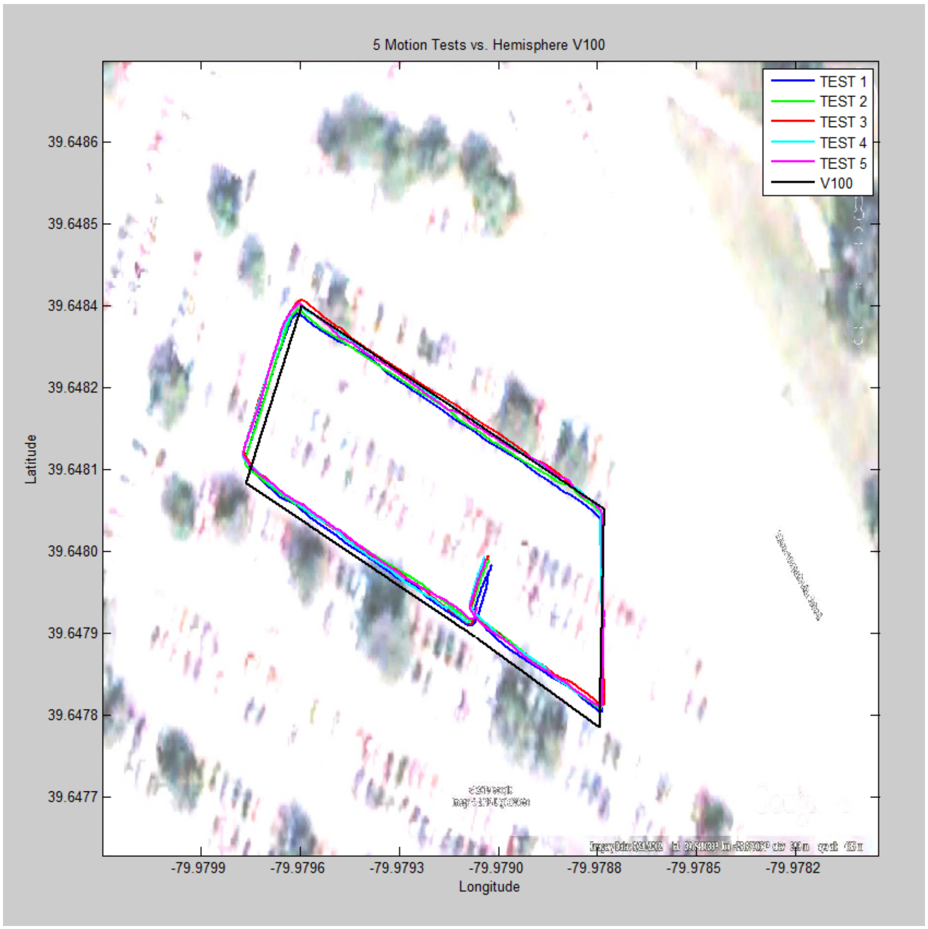


Figure 5-13: Position Estimate Comparison of Five NAS Tests and Hemisphere V100

In order to better determine the performance and mitigate the uncertainty injected by walking the course five different times, the average of the five traversals is plotted against the averaged NAS data in Figure 5-14. This figure shows that, on the course between corners one through three, the NAS positions were within 1 meter, averaging 0.9 meters with a standard deviation of 0.3 meters. During this portion of the course the heading data was accurate. The error increases to 3 meters in the worst case, from corner three back to the start of the path. The heading error was approximately 20 degrees following the turn at corner 3 and cause the inertial and GPS path's to diverge.

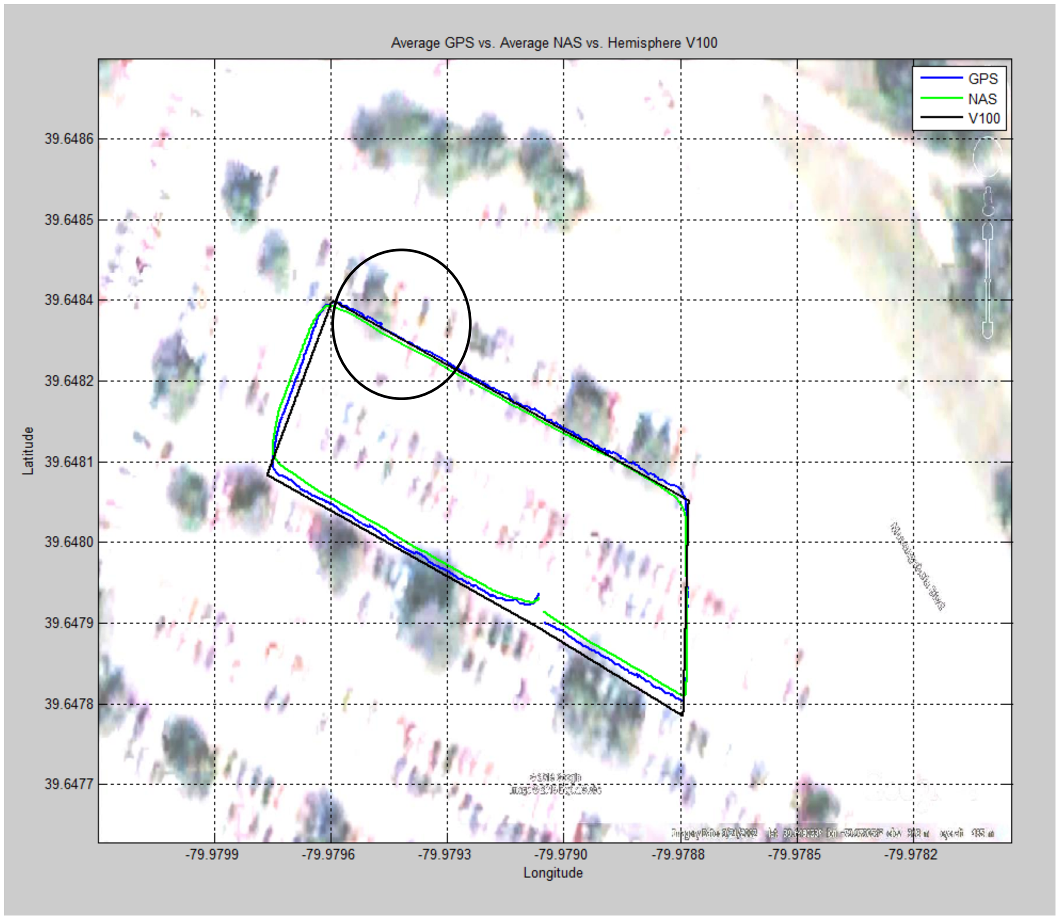


Figure 5-14: Position Comparison of Averaged NAS, GPS, and Hemisphere V100

Figure 5-15 shows a close up of the circled area in Figure 5-14. This plot demonstrates the nature of the improvement provided by the NAS' EKF. The EKF provides a better estimate of the path travelled. The raw GPS is within 2 meters of the V100 position, and within the GPS manufacturer's accuracy specification, however the noise in the receiver causes the position estimate to be less accurate than that of the EKF.

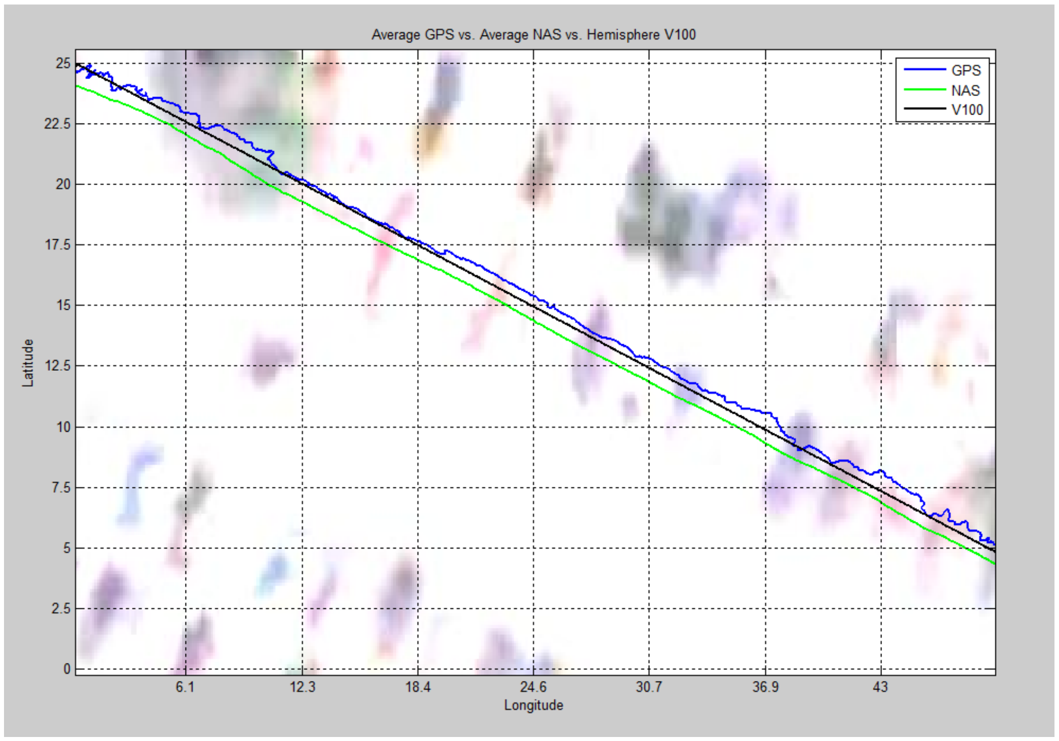


Figure 5-15: Position Comparison of Averaged NAS, GPS, and Hemisphere V100 (Close-up)

These plots show that the NAS provides position data that is comparable to the V100, and suitable for autonomous navigation purposes, where the goal is to reach a point within a 2 meter radius of the provided coordinates.

5.1.1.4. Loss of GPS Signal Testing and Results

The final test was to determine the usefulness of the NAS position data in the case of degraded or lost GPS signal. This test was performed by traversing the same path in the normal position traversal test with the modification of disconnecting the GPS antenna cable from the receiver board approximately 10m before reaching Corner 3, then reconnecting the cable approximately 10 meters afterwards. The results are shown in Figure 5-16. The circled area represents the time that the signal was lost.

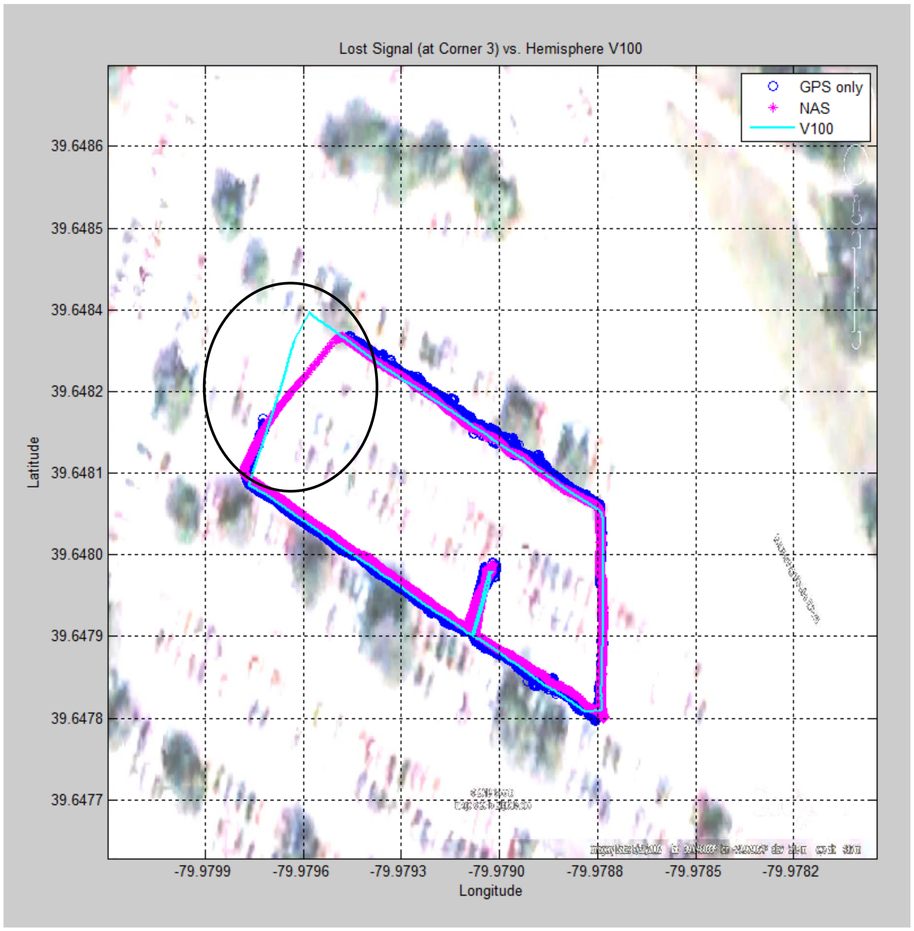


Figure 5-16: Position Comparison of NAS with lost signal and Hemisphere V100 (interpolated)

A close-up of the time that GPS is lost is shown in Figure 5-17. It is important to note that the V100 did not lose signal, and its path is shown to provide the truth data for the NAS. The position is clearly degraded; however the position remains within approximately 6 meters of expected position, and recovers quickly when the GPS signal is reacquired. Currently the NAS has no indication of the quality of the GPS signal, and it is expected that an algorithm that considers the loss of signal and ignores the measurement portion would improve this estimation. This is shown by the bias of individual points to the locations where GPS is received in Figure 5-17. The GPS receiver is capable of reporting Horizontal Dilution of Precision (HDOP) metrics that indicate the accuracy of the signal, however the GPS driver currently does not read these metrics in a trade-off to increase GPS signal throughput.

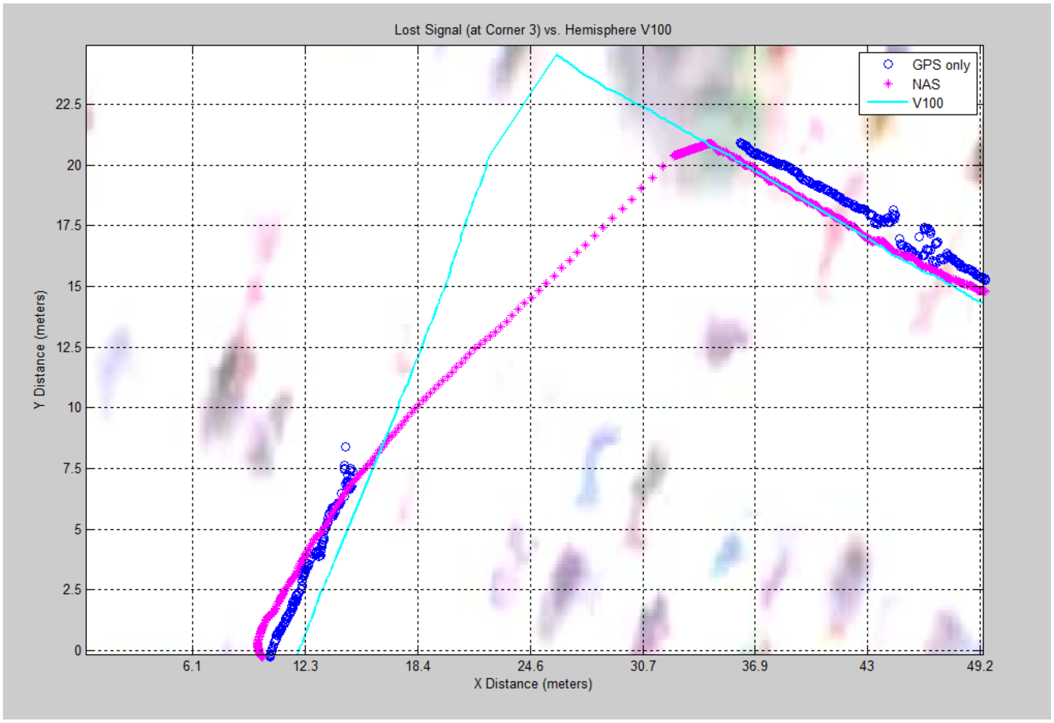


Figure 5-17: Position Comparison of NAS with lost GPS and Hemisphere V100 (close-up)

Chapter 6
Conclusions and Future Work

6.1. Conclusions

A literature review was conducted that studied rover design for challenging environments. This research focused on the NASA-inspired chassis design to handle a multitude of surfaces including large obstacles, loose sandy terrain, and steep inclines. Further research was performed on rover state estimation using inertial and GPS data fused via an EKF. Finally, research was performed on autonomous navigation, the intended future use of the rover state estimation.

A rover was designed, developed, fabricated, and ultimately tested at NASA's Johnson Space Center (JSC) Rock Yard in the 2012 RASC-AL RoboOps competition. WVU's team considered mission requirements set forth by the competition and the environment in which the rover was to operate in when designing the rover. Furthermore, the design strived to minimize mass, provide ample power reserves to handle a one hour mission, and be operable by a team stationed over 1,300 miles away from JSC at the WVU campus. The team placed fourth in the competition, despite a motor controller failure early in its trial.

A navigation assistance system was developed and demonstrated. This system utilized an IMU and GPS to determine three-axis rover attitude, velocity, and position states. This system was tested comparatively with a high precision DGPS unit and was shown to provide comparable results. With respect to position the NAS was able to provide accurate positions that averaged an error distance within 0.9 meters of the Hemisphere V100 DGPS receiver in the nominal case. In the worst case while still receiving GPS data the positions reported were within 3 meters of the V100-reported position. Further testing was performed to characterize the effect of the inertial portion of the EKF during GPS signal loss, and it was found that positions stayed within 6 meters of the V100 positions. However, analysis of the GPS data that was available to the EKF in this system implementation showed that performing a trade-off between data throughput and providing more data form the GPS receiver, namely HDOP data, would allow the EKF to adjust for loss of signal on the fly and transition to an inertial-only navigation system during times of signal loss.

Heading performance between the NAS and the Hemisphere V100 was also analyzed. It was observed that the heading was typically within 5 degrees of the V100, however when the system was facing south-west the heading error increased to nearly 20 degrees. The test was carried out in an area with cars, steel light poles and other possible sources of magnetic interference that may have contributed to this error. This error propagated into the position increasing the error from less than 1 meter to nearly 2 meters. Creating an accurate compass from magnetometers is a challenging endeavor. There are COTS solutions available with on-board filtering, temperature compensation, and advanced calibration techniques that may be able to remedy this issue in future work.

One persistent obstacle to this work was electronic interference. The complex set of electronics necessary to implement the rover created interference that degraded GPS signal to the point that GPS data was not usable in the NAS. In order to develop the algorithms necessary to implement the NAS, the system was removed and isolated from the rover in order to mitigate this interference.

Overall, it was observed that the Extended Kalman filter is capable of fusing data from different, but complimentary systems to form an optimized estimate of the rover's state. However, given the accuracy of the GPS used and the susceptibility to noise and errors on the inertial systems, it is determined that the rover may benefit from using the GPS alone. The accuracy of the GPS alone was comparable to the EKF and in some cases exceeded it. Also, the EKF can be costly on valuable processing resources.

The errors that persisted through this work pertained to individual systems, namely the lack of GPS HDOP feedback and magnetometer-based compass heading measurement. Further individualized study into these systems has the potential to improve this work.

6.2. Future Work

The research performed during the literature review explored potential applications of the work presented in this thesis. The following sections describe some potential improvements and applications of this work.

6.2.1. Interference Mitigation

One of the major, unforeseen, challenges to this research was the effect of electrical interference, caused by electronics on-board the rover, on the GPS receiver's ability to receive GPS signal and determine its position. The precise source of this interference is unknown, and could be any combination of the RCU computer, brushless motors, brushless motor controllers, and communications hardware. In order to reduce the mass of the rover, many of these electronics are not enclosed in metal cases that can reduce their electromagnetic emissions. A study to determine the cause of interference and a mitigation strategy would greatly benefit future use of this work.

6.2.2. Additional Sources of Data for Rover State

This research demonstrated that the NAS' combination of inertial sensors and GPS data through an EKF could provide position estimates that were comparable with the Hemisphere V100 high-precision COTS GPS unit. However, there may be room for improvement by considering alternative data. The rover offers both motor velocity and position counts that can be used to determine wheel velocity and distance-traveled, respectively. This data may provide a comparable or improved state prediction in the EKF.

The heading accuracy of the NAS was shown to be less than that of the Hemisphere V100, especially during motion, however the position was comparable. During motion, a course-over-ground algorithm could be implemented in order to better estimate the heading. Course-over-ground determines the heading by determining the angle between the current location and previous location.

6.2.3. Improvement of State Estimation with Degraded GPS

The NAS position was tested with the GPS antenna disconnected for a short time, preventing it from receiving GPS position updates and relying solely on inertial data in Section 5.1.1.4. This demonstrated that the position was able to stay within 6 meters of the actual path and recover quickly; however the 6 meter error is not accurate enough to provide good position feedback to an operator or autonomous navigation controller. The error observed can be traced to noisy accelerometer data as discussed in Chapter 4 as well as the NAS using the last received position as the measurement input, creating error. While the EKF adjusts its gain based on the error between the measurement (which is invalid) and the prediction, it still has some residual error from the invalid measurement. Currently, the GPS interface implemented in the NetBurner firmware is incapable of providing GPS-quality data to the NAS, thus the NAS is unaware of, and cannot adjust for, loss of GPS signal. Improving the GPS interface to provide additional data (at the potential cost of increasing serial bandwidth requirements) could provide a cue to the NAS to ignore the GPS measurements, and use alternative data to estimate the rover position.

6.2.4. Autonomous Navigation

An original goal of this research was to use data from the NAS to support autonomous navigation. This section proposes a simple autonomous \navigation algorithm that allows the operator to specify a set of Geodetic coordinates (latitude and longitude) for the rover to autonomously traverse to. Due to the rover operating in a relatively small area, the flat-Earth assumption is used to minimize calculation complexity. Geodetic coordinates are used for their ease of understandability, simple calculations of range and heading, and easy integration to mapping software such as Google Earth. The algorithm, illustrated in Figure 6-1, takes an operator-input goal position, calculates the heading and range to it, and then polls the EKF state output to determine the path to the goal and status of the traversal.

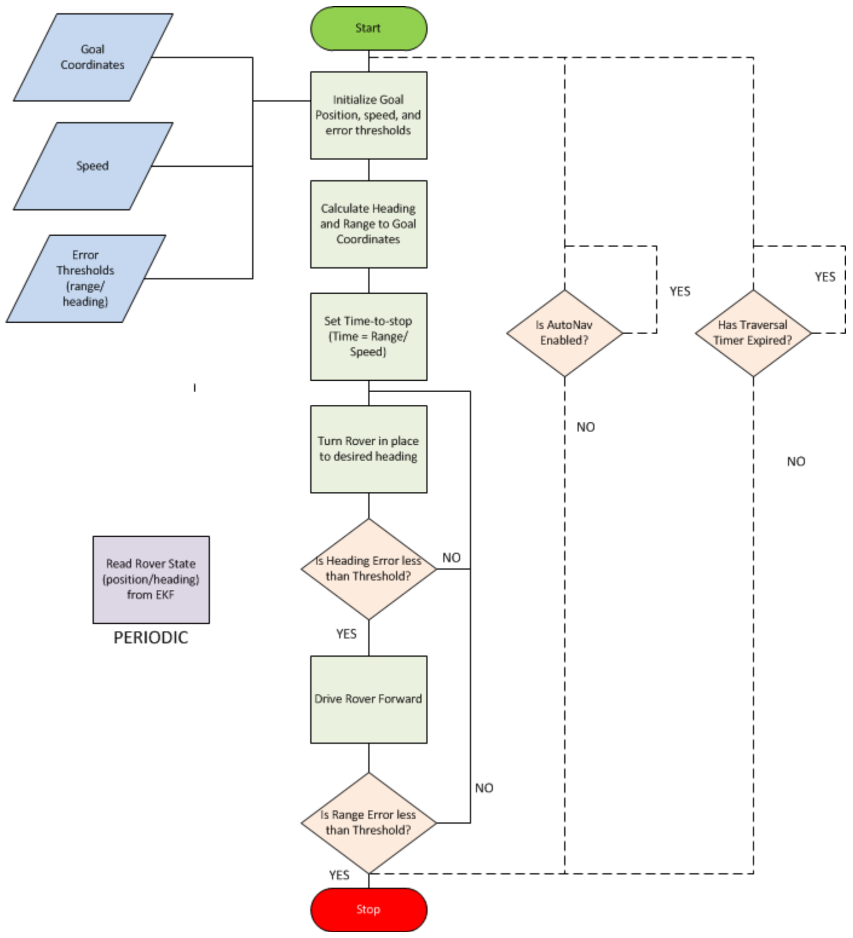

Goal Coordinates
Speed
Error Thresholds (range/ heading)
Start
Initialize Goal Position, speed, and error thresholds
Calculate Heading and Range to Goal Coordinates
Set Time-to-stop (Time = Range/ Speed)
Turn Rover in place to desired heading
Is Heading Error less than Threshold?
NO
YES
Drive Rover Forward
Is Range Error less than Threshold?
NO
YES
Stop
Read Rover State (position/heading) from EKF
PERIODIC
Is AutoNav Enabled?
YES
NO
Has Traversal Timer Expired?
YES
NO

Figure 6-1: Autonomous Navigation Control Flowchart

The traversal ends when the rover has reached an area near the goal within the specified error threshold. The autonomous navigation process employs the same fault protection as normal teleoperation, where any failure of communications will cause the rover to immediately stop itself. Furthermore, the operator can stop the rover at any time by either pressing the stop button on the OCU or manually controlling the rover. The calculated range and traversal speed are used to estimate the time it should take the rover to make the traversal. If the rover has not reached the goal in this time, it is stopped and the operator is notified.

This primitive algorithm forms a starting point for future work in autonomous navigation. The research presented in Chapter 2 details some successful autonomous navigation projects that used a combination of state estimation tools, such as the NAS, and obstacle sensors and avoidance algorithms.

Further research and development in this field could be applied to reduce the amount of teleoperation necessary to control the rover in future competitions.

References

[1] NASA/NIA, "2012 RASC-AL RoboOps Planetary Rover Design Requirements," NASA/NIA, [Online]. Available: http://www.nianet.org/RoboOps-2012/Program-Overview/2012-Planetary-Rover-Design-Requirements.aspx. [Accessed 29 April 2012].

[2] C. J. Voorhees and R. A. Lindemann, "Mars Exploration Rover Mobility Assembly Design, Test and Performance," *IEEE International Confereence on Systems, Man and Cybernetics,* vol. Vol 1., 2005.

[3] M. McKee, “Mars rover's broken wheel is beyond repair,” April 2006. [Online]. Available: http://www.newscientist.com/article/dn8944-mars-rovers-broken-wheel-is-beyond-repair.html.

[4] N. Wolchover, "NASA Gives Up On Stuck Mars Rover Spirit," 24 May 2011. [Online]. Available: http://www.space.com/11773-nasa-mars-rover-spirit-mission-ends.html.

[5] J. Biesiadecki, E. Tunstel, Y. Cheng and C. Leger, "Surface Navigation and Mobility Intelligence on the Mars Exploration Rovers," in *Intelligence for Space Robotics*, San Antonio, TSI Press, 2006, pp. 45-69.

[6] "MSL Curiosity Rover: Data Rates/Returns," [Online]. Available: http://mars.jpl.nasa.gov/msl/mission/communicationwithearth/data/. [Accessed 11 October 2012].

[7] J. J. Biesiadecki and M. W. Maimone, "The Mars Exploration Rover Surface Mobility Flight Software: Driving Ambition," *2006 IEEE Aerospace Conference Proceedings,* 2006.

[8] M. Bajracharya, M. W. Maimone and D. Helmick, "Autonomy for Mars Rovers: Past, Present, and Future," *Computer,* vol. 41, no. 12, pp. 44-50, 2008.

[9] Y. Cheng, M. Maimone and L. Matthies, "Two Years of Visual Odometry on the Mars Exploration Rovers," *J.Field Robotics,* pp. 169-186, 2007.

[10] S. Russell, "DARPA Grand Challenge Winner: Stanley the Robot!," *Popular Mechanics,* 9 January 2006.

[11] S. Thrun, M. Montemerio, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffman, K. Lau, C. Oakley, M. Palatucci, V. Pratt and P. Stang, "Stanley: The Robot that Won the DARPA Grand Challenge," *Journal of Field Robotics,* vol. 23, no. 9, pp. 661-692, 2006.

[12] J. Porter, "New Acquisition: Robo-car enters Smithsonian Collection," 2 November 2009. [Online]. Available: http://smithsonianscience.org/2009/11/robo-car-enters-smithsonian-collection/. [Accessed 23 October 2012].

[13] M. S. Grewal, L. R. Weill and A. P. Andrews, Global Positioning Systems, Inertial Navigation, and Integration, 2nd ed., Hoboken: John Wiley and Sons, 2007.

[14] S. F. Schmidt and L. A. McGee, "Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry," NASA, 1985.

[15] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME-Journal of Basic Engineering,* vol. 82, no. March 1960, pp. 35-45, 1960.

[16] P. Kim, Kalman Filter for Beginners with MATLAB Examples, Republic of Korea: A-JIN, 2011.

[17] D. Simon, Optimal State Estimation: Kalman, H-infinity and Nonlinear Approaches, Hoboken: John Wiley and Sons, Inc., 2006.

[18] Y. Gu, J. Gross, F. Barchesky, H. Chao and M. Napolitano, "Avionics Design for a Sub-Scale Fault Tolerant Flight Control Test Bed," in *Recent Advances in Aircraft Technology* , InTech, 2012.

[19] W. Li and J. Wang, "Effective Adaptive Kalman Filter for MEMS-IMU/Magnetometers Integrated Attitude and Heading Reference Systems," *Journal of Navigation,* pp. 1-15, 2012.

[20] RoboteQ, 4 August 2011. [Online]. Available: http://www.roboteq.com/files_n_images/files/datasheets/hbl2350datasheet.pdf.

[21] Phidgets Inc., "Phidgets 1001_0 Product Manual," 31 7 2008. [Online]. Available: http://www.phidgets.com/documentation/Phidgets/1001_0_Product_Manual.pdf. [Accessed 30 March 2013].

[22] Axis Communications, "Technical specifications – AXIS 215 PTZ-E Network Camera," 2008. [Online]. Available: http://www.axis.com/files/datasheet/ds_215ptz-e_33812_en_0812_lo.pdf. [Accessed 10 March 25].

[23] "AX-18F/AX-18A Technical Refernce Guide," Crustcrawler Robotics Website, [Online]. Available: http://www.crustcrawler.com/motors/AX-18F/docs/AX-18A%20Actuator%20Technical%20Reference%20Guide.pdf. [Accessed 10 May 2012].

[24] “DCDC-USB Advanced USN Configuration Manual,” [Online]. Available: http://resources.mini-box.com/online/PWR-DCDC-USB/PWR-DCDC-USB-Advanced-USB-configuration-manual.pdf. [Accessed 29 April 2012].

[25] "Crustcrawler Robotics AX-18A Smart Robotic Arm," Crustcrawler Robotics Website, 2011. [Online]. Available: http://www.crustcrawler.com/products/AX-18F%20Smart%20Robotic%20Arm/. [Accessed 10 May 2012].

[26] “Intel® Atom™ Processor N2800 Specifications,” Intel, [Online]. Available: http://ark.intel.com/products/58917/Intel-Atom-Processor-N2800-(1M-Cache-1_86-GHz). [Accessed 30 April 2012].

[27] "Intel Desktop Board DN2800MT Technical Product Specification," Intel, April 2012. [Online]. Available: http://downloadmirror.intel.com/20714/eng/DN2800MT_TechProdSpec02.pdf.

[28] RoboteQ, RoboteQ, 8 January 2011. [Online]. Available: http://www.roboteq.com/files_n_images/files/manuals/nxtgen_controllers_userman.pdf.

[29] "Validating the Reliability of Intel Solid State Drives," Intel, July 2011. [Online]. Available: http://www.intel.com/content/www/us/en/it-management/intel-it/intel-it-validating-reliability-of-

intel-solid-state-drives-brief.html. [Accessed April 2012].

[30] “Free Spirit: Follow Spirit's progress,” [Online]. Available: http://marsrovers.jpl.nasa.gov/newsroom/free-spirit.html. [Accessed 29 April 2012].

[31] "AForge .NET Framework," [Online]. Available: http://www.aforgenet.com/framework/. [Accessed 17 April 2012].

[32] "2012 Planetary Rover Design Requirements," NASA/NIA, [Online]. Available: http://www.nianet.org/RoboOps-2012/Program-Overview/Planetary-Rover-Design-Requirements.aspx. [Accessed 29 April 2012].

[33] Analog Devices, Inc., "Triaxial Inertial Sensor with Magnetometer ADIS16400/ADIS16405," 2009. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADIS16400_16405.pdf.

[34] NovAtel Inc., "OEM 615 Technical Data Sheet," July 2012. [Online]. Available: http://www.novatel.com/assets/Documents/Papers/OEM615.pdf. [Accessed 7 March 2013].

[35] NetBurner, Inc., "NetBurner, Inc.," 17 July 2012. [Online]. Available: http://www.netburner.com/download/Datasheet-MOD5213-100IR.pdf. [Accessed 27 March 2013].

[36] NovAtel Inc., "ANT-26C1GA-TBW-N and ANT-26C1GA-TBW-AN Antennas Guide," 26 August 2009. [Online]. Available: http://www.novatel.com/assets/Documents/Manuals/om-20000114.pdf. [Accessed 7 March 2013].

[37] R. Chow, "Evaluating Inertial Measurement Units," November 2011. [Online]. Available: http://www.tmworld.com/file/26170-TMW_1111_F4_IMU.pdf. [Accessed 30 April 2013].

[38] G. Cai, B. M. Chen and T. H. Lee, Unmanned Rotorcraft Systems, New York: Springer, 2011.

[39] M. J. Caruso, *Applications of Magnetoresistive Sensors in Navigation Systems,* Plymouth: Honeywell, Inc., 1997.

[40] C. Konvalin, "Compensating for Tilt, Hard-Iron, and Soft Iron Effects," 1 December 2009. [Online]. Available: http://www.sensorsmag.com/sensors/motion-velocity-displacement/compensating-tilt-hard-iron-and-soft-iron-effects-6475. [Accessed 9 January 2013].

[41] Micorsoft Developer Network, "Math.Atan2 Method," [Online]. Available: http://msdn.microsoft.com/en-us/library/system.math.atan2.aspx. [Accessed 21 March 2013].

[42] NOAA, "Estimated Value of Magnetic Declination [Calculator]," [Online]. Available: http://www.ngdc.noaa.gov/geomag-web/#declination. [Accessed 9 January 2013].

[43] Hemisphere GPS, "Hemisphere GPS V100 Data Sheet," 2010. [Online]. Available: http://www.hemispheregps.com/Portals/2/literature/V100_Data_Sheet_WEB_0609.pdf. [Accessed 12 February 2013].

Appendix A. Listing of Source Code for NAS

The source code that implements the NAS is listed in this appendix. In order to develop the EKF, two projects were developed. The first, IMU_Acq, which is a standalone implementation of the NAS, capable of providing real-time data, logging the NAS output, logging data products used within the NAS such as residuals and Kalman gain, and logging raw sensor data to be played back in order to tune and improve the NAS. The second, IMU_Player, is capable of playing raw data logs from IMU_Acq and post processing data. This was especially helpful in tuning the EKF, as well as finding and correcting sources of algortihm error. The appendix is organized by project

A.1 IMU_Acq Project

The IMU_Acq project is a standalone NAS interface that collects and processes NAS data to be used for analysis. It also logs raw sensor data to be replayed with the IMU_Player project.

A.1.1 Source Code List

A.1.1.1 Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using mars.RCU.Telemetry;

namespace IMU_Acq
{
    class Program
    {
        static void Main(string[] args)
        {
            // define telemetry and its devices
            TelemetryEngine t = new TelemetryEngine();

            // start processing telemetry
            t.Activate();

            Console.WriteLine("Press a key to deactivate telemetry");
            while (!Console.KeyAvailable)
            {
                // wait for keypress
                Thread.Sleep(10);
            }
            // stop processing telemetry
            Console.ReadKey(true);
            t.Deactivate();

            Console.WriteLine("Press a key to re-activate telemetry...");
            while (!Console.KeyAvailable)
            {
                // wait for keypress
                Thread.Sleep(10);
            }
            Console.ReadKey(true);
            t.Activate();


            while (!Console.KeyAvailable)
            {
                // wait for keypress
                Thread.Sleep(10);
            }

            Console.ReadKey(true);
            // stop processing telemetry
            t.Deactivate();

        }
    }
}
```

A.1.1.2 IMU.cs

```
#define LOG_RAW_DATA
#define LOG_NAS_DATA
#define LOG_MAG_DATA
//#define SUBTRACT_GRAV

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.IO;
using System.Threading;
using System.Collections.Concurrent;
using mars.RCU.Telemetry;

namespace mars.RCU.Models
{
    public class IMU : IProvideTelemetry
    {
#if LOG_MAG_DATA
        private StreamWriter logfile;
        string filename = string.Format("MAGLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
#endif
#if LOG_RAW_DATA
        private StreamWriter RawDatalogfile;
        string Rawfilename = string.Format("RAWLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);

#endif
#if LOG_NAS_DATA
        private StreamWriter NASlogfile;
        string NASfilename = string.Format("NASrecorderLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
#endif
        #region IMU Data Types and Vars
        const double MAG_DECLINATION = -9.5;

        private SerialPort serial_port;
        private Thread FindIMUPacket_Thread;
        private Boolean NotFinished = true;
        private ConcurrentQueue<Byte> BufferQ;

        private const Byte PACKET_FIRST_BYTE = 0xaa;    // packet first byte
        private const Byte PACKET_SECOND_BYTE = 68;     // packet second byte
        private const Byte PACKET_THIRD_BYTE = 18;      // packet third byte
        private const Byte IMU_PACKET_SIZE = 71;
        private Byte[] last_packet;
        private Byte[] packet;


        // rotation matrix for gps/imu
        private Double[] R1 = new Double[16];   /**4x4 Rotation matrix obtained using GPSLocation.m file**/
        private Double[] R2 = new Double[16];   /***4x4 inverse rotation matrix**/


    //CONSTANTS
        //BK15Aug2012:  According to data manula, cal const for gyro is 0.05 deg/s -> convert to rads 0.000873.   also added mag
        private Double[] cal_const = new Double[10] { 3.33e-3, 3.33e-3, 3.33e-3, 0.000873, 0.000873, 0.000873, 0.5e-3, 0.5e-3, 0.5e-3, 0.14 };
    /*calibration const only for accleration and rate gyros*/
        private Double G = 9.80665;
        private double Fil_x, Fil_y, Fil_z;
        private double Ecef_x, Ecef_y, Ecef_z;

        private double imu_mag_X, imu_mag_Y, imu_mag_Z; //raw acc values for attitude calculation
        private double imu_acc_X, imu_acc_Y, imu_acc_Z; //raw acc values for heading calculation
        private double magXoffset, magYoffset, magZoffset;

        private double gfield;                              //gravitational field at rover location for calculating attitude
        private double AccRoll, AccPitch;  //Accelerometer only pitch/roll for measurement input to EKF
        private double MagHeading;  //MAgnetometer heading with tilt comp for mesuremetn input to EKF
        double sinpitch, cospitch, sinroll, cosroll, Xh, Yh;             //vars to calculate tilt compensated heading

        private short[] uncal_imu_data = new short[10];    /*uncalibrated IMU data*/
        private double[] engr_imu_data = new Double[10];        /*calibrated IMU data*/
        private double[] gps_data = new Double[6] { 0, 0, 0, 0, 0, 0 };
        private Boolean is_active = false;
        private int EventRate = 1000;
        private DateTime LastHeardFromIMU;
        private DateTime TimeEventLastRaised;

        //public data
        public double gps_no_imu_lat;
        public double gps_no_imu_lon;
        public double gps_no_imu_velocity_X;
        public double gps_no_imu_velocity_Y;
        public double gps_no_imu_velocity_Z;
        public double gps_imu_lat;
        public double gps_imu_lon;
        public double gps_imu_velocity_X;
        public double gps_imu_velocity_Y;
        public double gps_imu_velocity_Z;
        public double imu_roll;
        public double imu_pitch;
        public double imu_heading;
        public double imu_headingNOTILT;

        #endregion

        #region Public Functions

        /// <summary>
        /// Event Raised for New IMU Telemetry Data
        /// </summary>
        public event EventHandler<TelemetryEventArgs<Object>> TelemetryUpdated_Event;
        public event EventHandler<TelemetryEventArgs<Object>> TelemetryUpdated;
        //public event EventHandler<Telemetry.TelemetryEventArgs> TelemetryUpdated;

        public IMU(String ComPort, int BaudRate, int TelemetryRateMilliSec)
        {


            // populate initial conditions - these are for morgantown.   when in houston, provide tanmay with X,Y,Z ECEF
            // and he will generate new initial conditions to plug in here.   see comments later.
            // todo, add this to config file.

            //Hard Coding Hard Iron offsets from MATLAB calibration
            //need to set this internally....  later

                magXoffset = 0.2648;
                magYoffset = -0.2430;
                magZoffset = -0.2127;

           // magXoffset = 0.0398; //Taken at tilt, no bueno
            //magYoffset = 0.1004;

#if LOG_MAG_DATA
            if (!File.Exists(filename))
            {
               logfile = File.CreateText(filename);
            }
#endif
#if LOG_RAW_DATA
            if (!File.Exists(Rawfilename))
            {
               RawDatalogfile = File.CreateText(Rawfilename);
            }
#endif
#if LOG_NAS_DATA
            if (!File.Exists(NASfilename))
            {
               NASlogfile = File.CreateText(NASfilename);
               NASlogfile.WriteLine("Time,     gps_no_imu_lat,     gps_no_imu_lon,     gps_imu_lat,     gps_imu_lon,     gps_no_imu_velocity_X,
gps_imu_velocity_X,  gps_no_imu_velocity_Y,  gps_imu_velocity_Y,  gps_no_imu_velocity_Z,  gps_imu_velocity_Z,  imu_roll,  imu_pitch,
imu_heading, imu_headingNOTILT");
            }
#endif
            //1313 Montrose Backyard -  CORRECTED
            R1[0] = 0.111505018570193;
            R1[1] = -0.628023349132296;
            R1[2] = -0.770164465408730;
            R1[3] = -21003.6732229171;
            R1[4] = 0.984601274790237; //was wrong
            R1[5] = 0.174815130012935;
            R1[6] = 0;
            R1[7] = 0.00298616569489241;
            R1[8] = 0.134636401151770;
            R1[9] = -0.758304914439577;
            R1[10] = 0.637845354472136;
            R1[11] = -6369733.37545383;
            R1[12] = 0;
            R1[13] = 0;
            R1[14] = 0;
            R1[15] = 1;

            R2[0] = 0.111505018570193;
            R2[1] = 0.984601274790238;
            R2[2] = 0.134636401151770;
            R2[3] = 859939.990000000;
            R2[4] = -0.628023349132296;
            R2[5] = 0.174815130012935;
            R2[6] = -0.758304914439577;
            R2[7] = -4843390.92000000;
            R2[8] = -0.770164465408730;
            R2[9] = 0;
            R2[10] = 0.637845354472137;
            R2[11] = 4046728.56000000;
            R2[12] = 0;
            R2[13] = 0;
        R2[14] = 0;
        R2[15] = 1;

        // create the serial port
        serial_port = new SerialPort(ComPort, BaudRate);
        serial_port.RtsEnable = false;
        serial_port.DataBits = 8;
        serial_port.Parity = Parity.None;
        serial_port.StopBits = StopBits.One;
        serial_port.DataReceived += new SerialDataReceivedEventHandler(serial_port_DataReceived);
        serial_port.ErrorReceived += new SerialErrorReceivedEventHandler(serial_port_ErrorReceived);
        serial_port.ReceivedBytesThreshold = 75;    //maximize chances of getting a "good" 71 byte packet

        LastHeardFromIMU = DateTime.UtcNow;
        TimeEventLastRaised = DateTime.UtcNow;

        // set the update event time
        EventRate = TelemetryRateMilliSec;

        // create the find packet thread
        FindIMUPacket_Thread = new Thread(new ThreadStart(FindPacket_DoWork));
        FindIMUPacket_Thread.IsBackground = true;
    }

    /* todo - read imu config from file
     * public IMU(IMUConfiguration config)
    {

    }
     */

    /// <summary>
    /// Activate IMU Telemetry
    /// </summary>
    public void Activate_Telemetry()
    {
        if (!is_active)
        {
            // init imu matrices
            EKF_Tan.init();

            // generate the queue - also clears it
            BufferQ = new ConcurrentQueue<byte>();

            // close the serial port if it is open
            if (serial_port.IsOpen)
            {
                serial_port.Close();
            }

            // start the thread
            NotFinished = true;
            switch (FindIMUPacket_Thread.ThreadState)
            {
                case ThreadState.Background | ThreadState.Unstarted:
                    {
                        FindIMUPacket_Thread.Start();
                        break;
                    }
                case ThreadState.Stopped:
                    {
                        // create the find packet thread
                        FindIMUPacket_Thread = new Thread(new ThreadStart(FindPacket_DoWork));
                        FindIMUPacket_Thread.IsBackground = true;
                        FindIMUPacket_Thread.Start();
                        break;
                    }
            }
            // open the serial port & clear buffers
            try
            {
                serial_port.Open();
                if (serial_port.IsOpen)
                {
                    serial_port.DiscardInBuffer();
                    serial_port.DiscardOutBuffer();
                }

                // set that we are active
                is_active = true;
            }
            catch (Exception ex)
            {
                //Handle Exception
            }

        }
        else
        {
            //may not want to throw exception here?
            throw new Exception("Telemetry is already active! Cannot activate");
        }
    }


    /// <summary>
    /// Deactivate IMU Telemetry
    /// </summary>
    public void Deactivate_Telemetry()
    {
        if (is_active)
        {
            is_active = false;

            // close the serial port
            if (serial_port.IsOpen)
            {
                serial_port.Close();
            }

            // shutdown the thread
            NotFinished = false;
            if (FindIMUPacket_Thread.ThreadState == ThreadState.Running)
            {
                FindIMUPacket_Thread.Join();
            }
        }
        else
        {
            // may not want to throw exception here?
            throw new Exception("Telemetry is NOT active!  Cannot Deactivate");
        }
    }

    /// <summary>
    /// Returns Whether IMU Telemetry is Active
    /// </summary>
    public Boolean IsActive()
    {
        return is_active;
    }


    /// <summary>
    /// Sets the rate at which the Telemetry available event will fire
    /// If it is set to zero, then telemetry will fire when it's available
    /// Defaults to 1 second
    /// </summary>
```

```
        /// <param name="RateInMilliSeconds"></param>
        public void SetTelemetryRate(int RateInMilliSeconds)
        {
            EventRate = RateInMilliSeconds;
        }


        /// <summary>
        /// Returns the telemetry rate
        /// </summary>
        /// <returns></returns>
        public int GetTelemetryRate()
        {
            return EventRate;
        }

        #endregion

        #region Private Functions


        private void serial_port_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {

#if DEBUG_MSGS
//TimeSpan ts = DateTime.UtcNow.Subtract(LastHeardFromIMU);
//Console.WriteLine("TS = :{0:d}", ts.Milliseconds);
#endif
            //record that we heard from IMU - this is used in worker thread to ensure that thread is operating on fresh data
            LastHeardFromIMU = DateTime.UtcNow;


            SerialPort sp = (SerialPort)sender;
            int BytesAvail = sp.BytesToRead;    //copy to var asap in case it were to change

            byte[] data = new byte[BytesAvail];
            sp.Read(data, 0, BytesAvail);

            // add the serial port data to the queue
            try
            {
                // enqueue the serial port bytes
                lock (BufferQ)
                {
                    for (int i = 0; i < BytesAvail; i++)
                    {
                        BufferQ.Enqueue(data[i]);
                    }

                    // empty the queue if it's too big.   just blow it away - we dont want to backup
                    if (BufferQ.Count > (IMU_PACKET_SIZE * 10))   // 100 packets = 5 seconds of 20Hz telemetry as the cutoff chosen arbitrarily
                    {
                        // clear a concurrent queue by allocating a new one.   This is the fastest way.   We could also dequeue 71 bytes but since
                        // we dont care about losing data, i say we just dump it and grab the next fresh telemetry packet.

                        Console.WriteLine("Clearing Queue - Count @  " + BufferQ.Count.ToString());
                        BufferQ = new ConcurrentQueue<byte>();

                        //Console.WriteLine("INFO: IMU Queue Too Big!  Not clearing!");
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception: " + ex.Message + " in " + ex.TargetSite.ToString());
            }
        }

        private void serial_port_ErrorReceived(object sender, SerialErrorReceivedEventArgs e)
        {
}

private void FindPacket_DoWork()
{
   Byte b1;
   Byte b2;
   Byte b3;
   Boolean ErrorOccurred = false;

   while (NotFinished)
   {
      ErrorOccurred = false;
      Thread.Sleep(10);

      //make sure we heard from IMU via serial port within 10 seconds.   If not, clear the Queue.
      //this ensures that if the IMU stops sending or the RCU stops getting data, stale packets in the
      //Q will not be sent and misconstrued for being "fresh".
      lock (BufferQ)
      {
         TimeSpan ts = DateTime.UtcNow.Subtract(LastHeardFromIMU);
         if (ts.TotalSeconds >= 10)
         {
            BufferQ = new ConcurrentQueue<byte>();
            Console.WriteLine("WARNING: IMU on Serial Port Not Heard From in 10 Seconds.   Dumping Stale Data");
         }

         // only process Q if it contains at least one packet
         if (BufferQ.Count > IMU_PACKET_SIZE)
         {
            try
            {
               if (BufferQ.TryDequeue(out b1))
               {
                  if (b1 == PACKET_FIRST_BYTE)
                  {
                     if (BufferQ.TryDequeue(out b2))
                     {
                        if (b2 == PACKET_SECOND_BYTE)
                        {
                           if (BufferQ.TryDequeue(out b3))
                           {
                              if (b3 == PACKET_THIRD_BYTE)
                              {
                                 // always allocate new memory for packet - do not reuse old
                                 packet = new byte[IMU_PACKET_SIZE];

                                 // packet found - put all bytes into a packet array
                                 packet[0] = b1;
                                 packet[1] = b2;
                                 packet[2] = b3;

                                 //  grab the rest of the bytes to form a packet
                                 for (int i = 3; i < (IMU_PACKET_SIZE - 3); i++)
                                 {
                                    // form the packet
                                    if (BufferQ.TryDequeue(out packet[i]))
                                    {
                                       //tweak this as necessary

                                    }
                                    else
                                    {
                                       Console.WriteLine("WARNING: Packet Dequeue Error.   Dumping Packet");
                                       ErrorOccurred = true;
                                    }
                                 }

                                 //generate imu data and raise event
                                 if (!ErrorOccurred)
```

```
                                            {
                                                GenerateIMUData(packet);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Exception: " + ex.Message + " in " + ex.TargetSite.ToString());
                }
            }
        }
    }
}


/// <summary>
/// Generate the IMU data from the packet and raise the event with the deep-copy data
/// </summary>
/// <param name="packet"></param>
private void GenerateIMUData(Byte[] packet)
{
//          IMU_Data data = new IMU_Data();

    // one last verification that this is a real IMU packet - check first 3 bytes
    if ((packet[0] == PACKET_FIRST_BYTE) && (packet[1] == PACKET_SECOND_BYTE) && (packet[2] == 
PACKET_THIRD_BYTE))
    {
        // set old packet to this new packet - use deep copy
        last_packet = new byte[IMU_PACKET_SIZE];
        Array.Copy(packet, last_packet, IMU_PACKET_SIZE);

        // uncalibrated imu data
        for (int i = 0; i < 10; i++)
        {
            byte high = (Byte)(packet[3 + 2 * i]);
            byte low = (Byte)(packet[4 + 2 * i]);
            uncal_imu_data[i] = (short)((short)(packet[3 + 2 * i] << 8) + (Byte)(packet[4 + 2 * i])); /*assembling IMU packet uncalibrated*/
        }

        // calibrated imu data?  SZ
        //bumped to 9 to get mag data
        for (int i = 0; i < 9; i++)
        {
            if (i < 3)
            {
                engr_imu_data[i] = (double)uncal_imu_data[i] * cal_const[i] * G;     /**constructing engineering data for only acceleration and 
rate gyros, magnetometer excluded*/
            }
            else
            {
                // removed 9.8
                engr_imu_data[i] = (double)uncal_imu_data[i] * cal_const[i];     /**constructing engineering data for only acceleration and rate 
gyros, magnetometer excluded*/
            }
        }

        //Convention:  EAST-NORTH-DOWN (all positive), nose up = pitch positive, right-side down = roll positive, turn clockwise = 
heading/yaw positive
        //Flip signs of gyro Gy, Gz   Ax  (to match END and std attitude conventions)  and Mz (to match Honeywell equations
        engr_imu_data[4] *= -1;  //Flip Gy
        engr_imu_data[5] *= -1;  //Flip Gz
        engr_imu_data[0] *= -1;  //Flip Ax

        //Try the Freescale AN, flip Y and Z to match
        engr_imu_data[7] *= -1;  //Flip My
```

```
            engr_imu_data[8] *= -1;  //Flip Mz



            //copy raw IMU values to local vars that make more sense for tracking through eqns
            //imu_acc_X = engr_imu_data[0];
            imu_acc_X = engr_imu_data[0];  //Flipped to match END convention (north  positive)
            imu_acc_Y = engr_imu_data[1];
            imu_acc_Z = engr_imu_data[2];
            imu_mag_X = engr_imu_data[6];
            imu_mag_Y = engr_imu_data[7];
            imu_mag_Z = engr_imu_data[8];  //flipping sign to match Honeywell, for equations

            gfield = 9.80665;  //use same constant as the EKF does.

            //Calculate Accelerometer based pitch and roll

            //Both Freescal AN3461 and Kim have it this way, I think I buggered itup!
            AccPitch = Math.Asin(imu_acc_X / gfield);
            AccRoll = Math.Asin(-imu_acc_Y/(gfield*Math.Cos(AccPitch)));
            sinpitch = Math.Sin(EKF_Tan.x[7]);
            sinroll = Math.Sin(EKF_Tan.x[6]);
            cospitch = Math.Cos(EKF_Tan.x[7]);
            cosroll = Math.Cos(EKF_Tan.x[6]);

#if LOG_MAG_DATA
            //string sMAGreadings = string.Format("{0:S},{1:F3},{2:F3}", DateTime.UtcNow.TimeOfDay.ToString(), Xh, Yh);
            string   sMAGreadings   =   string.Format("{0:S},{1:F3},{2:F3},{3:F3}",   DateTime.UtcNow.TimeOfDay.ToString(),   imu_mag_X,
imu_mag_Y, imu_mag_Z);
            //Console.WriteLine(sMAGreadings);
            logfile.WriteLine(sMAGreadings);
#endif
            imu_mag_X -= magXoffset;
            imu_mag_Y -= magYoffset;
            imu_mag_Z -= magZoffset;


            Xh = imu_mag_X * cospitch + imu_mag_Y * sinroll * sinpitch + imu_mag_Z * cosroll * sinpitch;
            Yh = imu_mag_Z * sinroll - imu_mag_Y * cosroll;  //-Yh per Freescale
            //Yh = imu_mag_Y * cosroll + imu_mag_Z * sinroll;

            MagHeading = Math.Atan2(Yh, Xh);
            if (MagHeading < 0) MagHeading += 2 * Math.PI;


            imu_headingNOTILT = Math.Atan2(-imu_mag_Y, imu_mag_X) + MAG_DECLINATION;
            if (imu_headingNOTILT < 0) imu_headingNOTILT += 2 * Math.PI;
            imu_headingNOTILT = imu_headingNOTILT * 180.0 / Math.PI;  //Convert to degrees



            Console.WriteLine("MX: {0:F3}  MY: {1:F3}  MZ: {2:F3}  XH: {3:F3}  YH: {4:F3}", imu_mag_X, imu_mag_Y, imu_mag_Z, Xh,
Yh);



            // GPS data constructor
            gps_data[0] = BitConverter.ToDouble(packet, 23);    //x
            gps_data[1] = BitConverter.ToDouble(packet, 31);    //y
            gps_data[2] = BitConverter.ToDouble(packet, 39);    //z
            gps_data[3] = BitConverter.ToDouble(packet, 47);    //vx
            gps_data[4] = BitConverter.ToDouble(packet, 55);    //vy
            gps_data[5] = BitConverter.ToDouble(packet, 63);    //vz


            //Write Raw values to CSV File for post processing and EKF tuning


            // fill the packet with raw gps data, no imu
            ECEF2LLA.ECF2LLA(gps_data[0], gps_data[1], gps_data[2], ref gps_no_imu_lat, ref gps_no_imu_lon);
            gps_no_imu_velocity_X = gps_data[3];
            gps_no_imu_velocity_Y = gps_data[4];
            gps_no_imu_velocity_Z = gps_data[5];

#if LOG_RAW_DATA
            //write imu data
            for (int i = 0; i < 10; i++)
            {
               RawDatalogfile.Write(uncal_imu_data[i].ToString());
               RawDatalogfile.Write(",");
            }
            //write GPS data
            for (int i = 0; i < 6; i++)
            {
               RawDatalogfile.Write(gps_data[i].ToString());
               RawDatalogfile.Write(",");
            }
            RawDatalogfile.Write(gps_no_imu_lat.ToString());
            RawDatalogfile.Write(",");
            RawDatalogfile.Write(gps_no_imu_lon.ToString());

            RawDatalogfile.Write("\n");
#endif
            /***converting GPS data from ECEF to Local coordinate for which we need Rotation matrix (4X4) which can be generated using
GPSLocation.m file, R[0]=Rotation matrix(1,1),
            *R[1]=Rotation matrix(1,2),R[2]=Rotation matrix(1,3),R[3]=Rotation matrix(1,4),R[4]=Rotation matrix(2,1) and so on, local
coordinate=R*x and local velocity=R*v **********************/

            //USe rotation matrix to convert raw GPS
            //init to zero, change only if GPS has value
            Double X_local = 0.0;
            Double Y_local = 0.0;
            Double Z_local = 0.0;
            if (gps_data[0] + gps_data[1] + gps_data[2] != 0.0)
            {
               X_local = R1[0] * gps_data[0] + R1[1] * gps_data[1] + R1[2] * gps_data[2] + R1[3];    // SZ: what is X?  Actual value or method to
calculate in C.   Not matlab.
               Y_local = R1[4] * gps_data[0] + R1[5] * gps_data[1] + R1[6] * gps_data[2] + R1[7];    // SZ: what is y?  Actual value or method to
calculate in C.   Not matlab.
               Z_local = R1[8] * gps_data[0] + R1[9] * gps_data[1] + R1[10] * gps_data[2] + R1[11];    // SZ: what is Z?  Actual value for method
to calculate in C.  Not matlab.
            }


            //BK28AUG:  CONVERSION IS OK

            Double Vx_local = R1[0] * gps_data[3] + R1[1] * gps_data[4] + R1[2] * gps_data[5];
            Double Vy_local = R1[4] * gps_data[3] + R1[5] * gps_data[4] + R1[6] * gps_data[5];
            Double Vz_local = R1[8] * gps_data[3] + R1[9] * gps_data[4] + R1[10] * gps_data[5];

            /*********the above local coordinate is in South East Up the EKF assumes South West Down so below we convert SEU to SWD**/
            Double X_SWD = X_local;
            Double Y_SWD = -Y_local;     //SZ: Notice the minus sign
            Double Z_SWD = -Z_local;     //SZ: Notice the minus sign

            Double Vx_SWD = Vx_local;
            Double Vy_SWD = -Vy_local;   //SZ: Notice    the minus sign
            Double Vz_SWD = -Vz_local;   //SZ: Notice the minus sign

            //BK17SEP2012:  combining gps measurement with measured attitude from accel/mag.   May want to look at better storage of this
            double[] measurement = new double[] { X_SWD, Y_SWD, Z_SWD, Vx_SWD, Vy_SWD, Vz_SWD, AccRoll, AccPitch,
MagHeading };

            // after this function is called, simply referenced the x array from the EKF_Tan module.   this matrix x
            // contains the fused data.   Yes I know this is terrible way of getting the data, but it is what I have to work
            // with right now.
            EKF_Tan.ApplyFilter(engr_imu_data, measurement);

            //convert to ecf
            Fil_x = EKF_Tan.x[0];
```

```
                Fil_y = -EKF_Tan.x[1];
                Fil_z = -EKF_Tan.x[2];
                //Console.WriteLine("Fil_x: " + Fil_x.ToString());
                //Console.WriteLine("Fil_y: " + Fil_y.ToString());
                //Console.WriteLine("Fil_z: " + Fil_z.ToString());

                Ecef_x = R2[0] * Fil_x + R2[1] * Fil_y + R2[2] * Fil_z + R2[3];
                Ecef_y = R2[4] * Fil_x + R2[5] * Fil_y + R2[6] * Fil_z + R2[7];
                Ecef_z = R2[8] * Fil_x + R2[9] * Fil_y + R2[10] * Fil_z + R2[11];


                ECEF2LLA.ECF2LLA(Ecef_x, Ecef_y, Ecef_z, ref gps_imu_lat, ref gps_imu_lon);
                gps_imu_velocity_X = EKF_Tan.x[3];
                gps_imu_velocity_Y = EKF_Tan.x[4];
                gps_imu_velocity_Z = EKF_Tan.x[5];

                //21AUG2012BK:  Is Attitude stored in 6.7.8?
                imu_roll = EKF_Tan.x[6] * 180.0 / Math.PI;
                imu_pitch = EKF_Tan.x[7] * 180.0 / Math.PI;
                imu_heading = EKF_Tan.x[8] * 180.0 / Math.PI;


                String sIMUtelemetry = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9},
{8:F9},{9:F9},{10:F9},{11:F9},{12:F9},{13:F9},{14:F9}", DateTime.UtcNow.TimeOfDay.ToString(), gps_no_imu_lat, gps_no_imu_lon,
gps_imu_lat, gps_imu_lon, gps_no_imu_velocity_X, gps_imu_velocity_X, gps_no_imu_velocity_Y, gps_imu_velocity_Y,
gps_no_imu_velocity_Z, gps_imu_velocity_Z, imu_roll, imu_pitch, imu_heading, imu_headingNOTILT);
                Console.WriteLine(sIMUtelemetry);
                */
                NASlogfile.WriteLine(sIMUtelemetry);


            }
            else
            {
                Console.WriteLine("INFO: Packet did not pass final validation!");
            }
        }
        #endregion
    }

}
```

A.1.1.3 EKF.cs

```
#define LOG_QR  //Define if you want to log mearuremnt data to determine variance
#define LOG_RES //Log residuals to see if converging
#define LOG_K  //Log Kalman Gain
//#define ZERO_VY_VZ  //drop the laws of ground vehicle physics on it

using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;

namespace mars.RCU.Models
{
    public class EKF
    {
        public const double PI = 3.141592653589793;
        public const int SPIKE = 600;

        #region Variables


#if LOG_QR
        //Logs to determine variance of inputs to EKF
        private static StreamWriter Qlogfile;// = new StreamWriter();
        private static StreamWriter Rlogfile;// = new StreamWriter();
```

```
	private static string QFilename = string.Format("Q_LOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
	private static string RFilename = string.Format("R_LOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);

#endif
#if LOG_RES
	//Logs to determine variance of inputs to EKF
	private static StreamWriter Reslogfile;// = new StreamWriter();
	private static string ResFilename = string.Format("RES_LOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);

#endif
#if LOG_K
	//Logs to determine variance of inputs to EKF
	private static StreamWriter Klogfile;// = new StreamWriter();
	private static string KFilename = string.Format("GAIN_LOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);

#endif
	public static double[] um1 = new double[12];
	// local Variable Definition
	public static double S_Phi;
	public static double C_Phi;
	public static double S_Theta;
	public static double C_Theta;
	public static double T_Theta;
	public static double Sec_Theta;
	public static double S_Psi;
	public static double C_Psi;
	//public static double Ts = 0.1;  //Data taken at 10 Hz, Ts = 100 ms   MATLAB ANALYSIS MAKES THIS APPEAR TO BE 20 Hz,
WHY?!?!?!?
	public static double Ts = 0.05;  //Data taken at 10 Hz, Ts = 100 ms   MATLAB ANALYSIS MAKES THIS APPEAR TO BE 20 Hz,
WHY?!?!?!?
	public static double G = 9.80665;
	public static double t;

	public static double[] A = new double[81];
	public static double[] AT = new double[81];
	public static double[] temp = new double[81];
	public static double[] temp1 = new double[81];
	public static double[] temp2 = new double[81]; //PHT
	public static double[] temp3 = new double[81]; //HP
	public static double[] temp4 = new double[81]; //HPHT
	public static double[] rwv = new double[181]; //HPHT+R  //       public static double[] temp6 = new double[36]; //inv(HPHT+R)
	public static double[] temp6 = new double[81]; //inv(HPHT+R)
	public static double[] xm = new double[9]; // INS Predicted states
	public static double[] xK = new double[9];
	public static double[] PK = new double[81];
	public static double[] Res = new double[9]; // Difference  of Predicted State and measnurment
	public static double[] Pm = new double[81]; // INS Predicted Covariance
	public static double[] xUP = new double[9]; // INS Predicted states
	public static double[] PUP = new double[81]; // INS Predicted Covariance
	public static double[] x = new double[9];
	public static double[] p = new double[81];


	public static double Rate_Diff = 1; //Rate Difference Between GPS and INS
	public static int i;
	public static int j;
	public static int k;
	public static int l;
	public static int m;
	public static double[] Q = new double[81]; //Process Noise Covariance Matrix

	public static double[] R = new double[81]; // Measurment Noise Covariance Matrix
	public static double[] H = new double[81]; // Observation matrix
	public static double[] HT = new double[81]; //Transpose of Observation Matrix
	public static double[] K = new double[81];
	public static double[] KH = new double[81];
```

```
    public static double[] IKH = new double[81];
    public static double[] Kef = new double[9]; //The actual addition K adds to the predicted states

    //Adding history variables to show prediction integration error, if exists
    private static double[] xplast = new double[9];
    //Tuning Notes
    //Increasing Q:
    //Increasing R:



    //public static double[] diagQ = { 0, 0, 0, 0.149529501, 0.147740194, 0.04058798, 0.03306099, 0.034122935, 0.039941555 }; ///Taken
experimentally, these are integrated Need to account for time?
    //TUNING 26 MARCH:  Above is way too low, not indicative of noise
    //Increasing attitude(gyro) noise only, and started accel
    //public static double[] diagQ = { 0, 0, 0, 0.2, 0.2, 0.2, 0.3, 0.3, 1.1 };
    public static double[] diagQ = { 0, 0, 0, 0.3, 0.3, 0.3, 0.003, 0.003, 0.07};

    public static double ALPHA = 1;  //Lower Alpha: Less Measurement in answer, Higher Alpha: More measurement in answer


    //Tuning, Leaving Attitude alone for now, trying velocity
    public static double[] diagR = { 3, 3, 3, ALPHA * 0.3, ALPHA * 0.3, ALPHA * 0.3, ALPHA * 0.005, ALPHA * 0.005, ALPHA * 0.05 };


    public static int[] SPIKE_FLAG = { 0, 0, 0, 0, 0, 0 };
    public static int[] IMU_SPIKE_FLAG = { 0, 0, 0, 0, 0, 0 };

    //BK24Feb13:  Break firstrun flag up to allow IMU to function indoors w/o GPS for development

    private static Boolean IMUFirstRun = true; //flag to initialize EKF with initial IMU values
    private static Boolean GPSFirstRun = true; //flag to initialize EKF with initial GPS values (once valid)
    //Send out the count
    public static double count = 0;

    #endregion

    public static double pythag2(double a, double b)
    {
        double at = Math.Abs(a);
        double bt = Math.Abs(b);
        double ct;
        if (at > bt)
        {
            ct = bt / at;
            return at * Math.Sqrt(1.0 + ct * ct);
        }
        else
        {
            if (bt != 0)
            {
                ct = at / bt;
                return bt * Math.Sqrt(1.0 + ct * ct);
            }
            else
                return 0;
        }
    }


    public static void svdcmp2(double[] rwv, int m, int n)
    {
        int flag;
        int i;
        int its;
        int j;
        int jj;
        int k;
        int l = 0;
        int nm = 0;
```

```
double c;
double f;
double h;
double s;
double x;
double y;
double z;
double anorm = 0.0;
double g = 0.0;
double scale = 0.0;

for (i = 1; i <= n; i++)
{
   l = i + 1;
   rwv[n * (m + n + 1) - 1 + i] = scale * g;
   g = s = scale = 0.0;
   if (i <= m)
   {
      for (k = i; k <= m; k++)
         scale += Math.Abs(rwv[-1 + k - m + m * i]);
      if (scale != 0)
      {
         for (k = i; k <= m; k++)
         {
            rwv[-1 + k - m + m * i] /= scale;
            s += rwv[-1 + k - m + m * i] * rwv[-1 + k - m + m * i];
         }
         f = rwv[-1 + i - m + m * i];
         g = -((f) >= 0.0 ? Math.Abs(Math.Sqrt(s)) : -Math.Abs(Math.Sqrt(s)));
         h = f * g - s;
         rwv[-1 + i - m + m * i] = f - g;
         if (i != n)
         {
            for (j = l; j <= n; j++)
            {
               for (s = 0.0, k = i; k <= m; k++)
                  s += rwv[-1 + k - m + m * i] * rwv[-1 + k - m + m * j];
               f = s / h;
               for (k = i; k <= m; k++)
                  rwv[-1 + k - m + m * j] += f * rwv[-1 + k - m + m * i];
            }
         }
         for (k = i; k <= m; k++)
            rwv[-1 + k - m + m * i] *= scale;
      }
   }
   rwv[m * n - 1 + i] = scale * g;
   g = s = scale = 0.0;
   if (i <= m && i != n)
   {
      for (k = l; k <= n; k++)
         scale += Math.Abs(rwv[-1 + i - m + m * k]);
      if (scale != 0)
      {
         for (k = l; k <= n; k++)
         {
            rwv[-1 + i - m + m * k] /= scale;
            s += rwv[-1 + i - m + m * k] * rwv[-1 + i - m + m * k];
         }
         f = rwv[-1 + i - m + m * l];
         g = -((f) >= 0.0 ? Math.Abs(Math.Sqrt(s)) : -Math.Abs(Math.Sqrt(s)));
         h = f * g - s;
         rwv[-1 + i - m + m * l] = f - g;
         for (k = l; k <= n; k++)
            rwv[n * (m + n + 1) - 1 + k] = rwv[-1 + i - m + m * k] / h;
         if (i != m)
         {
            for (j = l; j <= m; j++)
            {
               for (s = 0.0, k = l; k <= n; k++)
```

```
                        s += rwv[-1 + j - m + m * k] * rwv[-1 + i - m + m * k];
                    for (k = l; k <= n; k++)
                        rwv[-1 + j - m + m * k] += s * rwv[n * (m + n + 1) - 1 + k];
                }
            }
            for (k = l; k <= n; k++)
                rwv[-1 + i - m + m * k] *= scale;
        }
    }
    anorm = Math.Max(anorm, (Math.Abs(rwv[m * n - 1 + i]) + Math.Abs(rwv[n * (m + n + 1) - 1 + i])));
}
for (i = n; i >= 1; i--)
{
    if (i < n)
    {
        if (g != 0)
        {
            for (j = l; j <= n; j++)
                rwv[n * (m + 1) - 1 + j - n + n * i] = (rwv[-1 + i - m + m * j] / rwv[-1 + i - m + m * l]) / g;
            for (j = l; j <= n; j++)
            {
                for (s = 0.0, k = l; k <= n; k++)
                    s += rwv[-1 + i - m + m * k] * rwv[n * (m + 1) - 1 + k - n + n * j];
                for (k = l; k <= n; k++)
                    rwv[n * (m + 1) - 1 + k - n + n * j] += s * rwv[n * (m + 1) - 1 + k - n + n * i];
            }
        }
        for (j = l; j <= n; j++)
            rwv[n * (m + 1) - 1 + i - n + n * j] = rwv[n * (m + 1) - 1 + j - n + n * i] = 0.0;
    }
    rwv[n * (m + 1) - 1 + i - n + n * i] = 1.0;
    g = rwv[n * (m + n + 1) - 1 + i];
    l = i;
}
for (i = n; i >= 1; i--)
{
    l = i + 1;
    g = rwv[m * n - 1 + i];
    if (i < n)
        for (j = l; j <= n; j++)
            rwv[-1 + i - m + m * j] = 0.0;
    if (g != 0)
    {
        g = 1.0 / g;
        if (i != n)
        {
            for (j = l; j <= n; j++)
            {
                for (s = 0.0, k = l; k <= m; k++)
                    s += rwv[-1 + k - m + m * i] * rwv[-1 + k - m + m * j];
                f = (s / rwv[-1 + i - m + m * i]) * g;
                for (k = i; k <= m; k++)
                    rwv[-1 + k - m + m * j] += f * rwv[-1 + k - m + m * i];
            }
        }
        for (j = i; j <= m; j++)
            rwv[-1 + j - m + m * i] *= g;
    }
    else
    {
        for (j = i; j <= m; j++)
            rwv[-1 + j - m + m * i] = 0.0;
    }
    ++rwv[-1 + i - m + m * i];
}
for (k = n; k >= 1; k--)
{
    for (its = 1; its <= 30; its++)
    {
        flag = 1;
```

```
for (l = k; l >= 1; l--)
{
    nm = l - 1;
    if ((double)(Math.Abs(rwv[n * (m + n + 1) - 1 + l]) + anorm) == anorm)
    {
        flag = 0;
        break;
    }
    if ((double)(Math.Abs(rwv[m * n - 1 + nm]) + anorm) == anorm)
        break;
}
if (flag != 0)
{
    c = 0.0;
    s = 1.0;
    for (i = l; i <= k; i++)
    {
        f = s * rwv[n * (m + n + 1) - 1 + i];
        rwv[n * (m + n + 1) - 1 + i] = c * rwv[n * (m + n + 1) - 1 + i];
        if ((double)(Math.Abs(f) + anorm) == anorm)
            break;
        g = rwv[m * n - 1 + i];
        h = EKF.pythag2(f, g);
        rwv[m * n - 1 + i] = h;
        h = 1.0 / h;
        c = g * h;
        s = (-f * h);
        for (j = 1; j <= m; j++)
        {
            y = rwv[-1 + j - m + m * nm];
            z = rwv[-1 + j - m + m * i];
            rwv[-1 + j - m + m * nm] = y * c + z * s;
            rwv[-1 + j - m + m * i] = z * c - y * s;
        }
    }
}
z = rwv[m * n - 1 + k];
if (l == k)
{
    if (z < 0.0)
    {
        rwv[m * n - 1 + k] = -z;
        for (j = 1; j <= n; j++)
            rwv[n * (m + 1) - 1 + j - n + n * k] = (-rwv[n * (m + 1) - 1 + j - n + n * k]);
    }
    break;
}
if (its == 50)
{
    Console.Write("No convergence in 50 SVDCMP2 iterations \n");
    return;
}

x = rwv[m * n - 1 + l];
nm = k - 1;
y = rwv[m * n - 1 + nm];
g = rwv[n * (m + n + 1) - 1 + nm];
h = rwv[n * (m + n + 1) - 1 + k];
f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0 * h * y);
g = EKF.pythag2(f, 1.0);
f = ((x - z) * (x + z) + h * ((y / (f + ((f) >= 0.0 ? Math.Abs(g) : -Math.Abs(g)))) - h)) / x;
c = s = 1.0;
for (j = l; j <= nm; j++)
{
    i = j + 1;
    g = rwv[n * (m + n + 1) - 1 + i];
    y = rwv[m * n - 1 + i];
    h = s * g;
    g = c * g;
    z = EKF.pythag2(f, h);
```

```
                        rwv[n * (m + n + 1) - 1 + j] = z;
                        c = f / z;
                        s = h / z;
                        f = x * c + g * s;
                        g = g * c - x * s;
                        h = y * s;
                        y = y * c;
                        for (jj = 1; jj <= n; jj++)
                        {
                            x = rwv[n * (m + 1) - 1 + jj - n + n * j];
                            z = rwv[n * (m + 1) - 1 + jj - n + n * i];
                            rwv[n * (m + 1) - 1 + jj - n + n * j] = x * c + z * s;
                            rwv[n * (m + 1) - 1 + jj - n + n * i] = z * c - x * s;
                        }
                        z = EKF.pythag2(f, h);
                        rwv[m * n - 1 + j] = z;
                        if (z != 0)
                        {
                            z = 1.0 / z;
                            c = f * z;
                            s = h * z;
                        }
                        f = (c * g) + (s * y);
                        x = (c * y) - (s * g);
                        for (jj = 1; jj <= m; jj++)
                        {
                            y = rwv[-1 + jj - m + m * j];
                            z = rwv[-1 + jj - m + m * i];
                            rwv[-1 + jj - m + m * j] = y * c + z * s;
                            rwv[-1 + jj - m + m * i] = z * c - y * s;
                        }
                    }
                    rwv[n * (m + n + 1) - 1 + l] = 0.0;
                    rwv[n * (m + n + 1) - 1 + k] = f;
                    rwv[m * n - 1 + k] = x;
                }
            }
        }

        public static void init()
        {
#if LOG_QR
            //Logs to determine variance of inputs to EKF

            //Open Q log file
            if (!File.Exists(QFilename))
            {
                Qlogfile = File.CreateText(QFilename);
            }
            //Open R log file
            if (!File.Exists(RFilename))
            {
                Rlogfile = File.CreateText(RFilename);
            }
#endif
#if LOG_RES
            if (!File.Exists(ResFilename))
            {
                Reslogfile = File.CreateText(ResFilename);
            }

#endif
#if LOG_K
            if (!File.Exists(KFilename))
            {
                Klogfile = File.CreateText(KFilename);
            }
#endif
```

```
            //Populate the Q Matrix
            for (l = 0; l < 9; l++) //Column Index
            {
                for (m = 0; m < 9; m++) //Row Index
                {
                    if (l == m)
                        Q[m + l * 9] = diagQ[l];
                    else
                        Q[m + l * 9] = 0;
                }
            }

            //Populate the R Matrix
            for (l = 0; l < 9; l++) //Column Index
            {
                for (m = 0; m < 9; m++) //Row Index
                {
                    if (l == m)
                        R[m + l * 9] = diagR[l];
                    else
                        R[m + l * 9] = 0;
                }
            }
            //Populate the H Matrix
            //BK17SEP2012:  No observation of roll, pitch, yaw...   They are just the [noisy] integral of gyro data
            //There may be room for improvement since they are used in velocity and indirectly by position prediction
            //Can use accelerometers as measurement of attiude to feed into residual calcualtion in EKF
            //Yaw can't be derived from accleerometers, however this may be the place to bring in mag heading, using it as the observation
            //for yaw.   Essentially I can use the derivtion of roll,pitch from accelerometers/gravity and the yaw from mag heading and
            //use them to calibrate noisy gyros per kalman for beginners, approx page 163
            for (l = 0; l < 9; l++) //Column Index
            {
                //for (m = 0; m < 6; m++) //Row Index
                for (m = 0; m < 9; m++) //Row Index
                {
                    if (l == m)
                        H[m + l * 9] = 1;
                    else
                        H[m + l * 9] = 0;
                }
            }

            //H Transpose
            k = 0;

            for (i = 0; i < 9; i++)
            {
                for (j = 0; j < 9; j++)
                {
                    HT[j + i * 9] = H[i + j * 9];
                }
            }
        }

        public static void ApplyFilter(double[] u, double[] z)
        {

    //Check for 1st run of filter, need initial values to seed EKF
        //Any value to averaging the initial values to get better start??
            //Seperate checks for IMU and GPS for indoor devleopment work [on IMU]
            if (IMUFirstRun)
            {
                //BK17SEP2012: Need initial states.   May want to move this out and do a check for GPS lock before filter start
                for (i = 3; i < 9; i++) //Just velocity and attitude
                {
                    x[i] = z[i];  //Seed the filter with first measurement data to eliminate spike throw out
                }
                IMUFirstRun = false;
            }
            //Check if GPS is valid
```

```
if (GPSFirstRun)
{
    if ((z[0] + z[1] + z[2]) != 0.0)  //XYZ position
    {
        for (i = 0; i < 6; i++)
        {
            x[i] = z[i];
        }
        GPSFirstRun = false;
    }
}

if (count > 0)
{

    for (i = 0; i < 6; i++)
    {
        if ((z[i] - x[i]) > 1000)
            SPIKE_FLAG[i] = 1;
        else
        {
            if ((z[i] - x[i]) < -(1000))
                SPIKE_FLAG[i] = 1;
            else
                SPIKE_FLAG[i] = 0;
        }
    }


    // Eliminate Large IMU SPIKES
    //rate gyro limit


    for (i = 0; i < 3; i++)
    {
        //if (u[i + 3] > (150 * (PI / 180)))  //BK24SEP Why convert to rads, think this is mistake
        if (u[i + 3] > 150)
            IMU_SPIKE_FLAG[i + 3] = 1;
        else
        {
            if (u[i + 3] < -150)
                IMU_SPIKE_FLAG[i + 3] = 1;
            else
                IMU_SPIKE_FLAG[i + 3] = 0;
        }

    }
    //accelerometer limit
    for (i = 0; i < 3; i++)
    {
        if (u[i] > (150))
            IMU_SPIKE_FLAG[i] = 1;
        else
        {
            if (u[i] < (-150))
                IMU_SPIKE_FLAG[i] = 1;
            else
                IMU_SPIKE_FLAG[i] = 0;
        }

    }
}

count++;
for (i = 0; i < 9; i++)
    xK[i] = x[i];
for (i = 0; i < 81; i++)
    PK[i] = p[i];
```

```
        // Precalculate Triginometric Values
        S_Phi = Math.Sin(xK[6]);
        C_Phi = Math.Cos(xK[6]);
        S_Theta = Math.Sin(xK[7]);
        C_Theta = Math.Cos(xK[7]);
        T_Theta = Math.Tan(xK[7]);
        Sec_Theta = 1 / Math.Cos(xK[7]);
        S_Psi = Math.Sin(xK[8]);
        C_Psi = Math.Cos(xK[8]);

        //BK19SEP2012 Zeroing out GPS velocity for bench test without GPS
        //z[3] = 0;
        //z[4] = 0;
        //z[5] = 0;

        //Only use new IMU data if no spike
        //BK17SEP2012  um is never used.   Need to decide if want to propogate z or use this check.
        if ((IMU_SPIKE_FLAG[0] + IMU_SPIKE_FLAG[1] + IMU_SPIKE_FLAG[2] + IMU_SPIKE_FLAG[3] + IMU_SPIKE_FLAG[4] +
IMU_SPIKE_FLAG[5]) == 0)
        {
          for (i = 0; i < 6; i++)
            um1[i] = u[i];
        }

        //Only use new GPS data if no spike
        if ((SPIKE_FLAG[0] + SPIKE_FLAG[1] + SPIKE_FLAG[2] + SPIKE_FLAG[3] + SPIKE_FLAG[4] + SPIKE_FLAG[5]) == 0)
        {
          for (i = 0; i < 6; i++)
            um1[i + 6] = z[i];
        }

        //printf("%d,%d\r\n",(int)G_FLAG,(int)I_FLAG);
        //printf("%d,%d\r\n",(int)I_FLAG,(int)G_FLAG);
        // SPIKE_FLAG [0] = 0;SPIKE_FLAG [1] = 0;SPIKE_FLAG [2] = 0;SPIKE_FLAG [3] = 0;SPIKE_FLAG [4] = 0;SPIKE_FLAG [5] =
0;
        //   IMU_SPIKE_FLAG[0] = 0; IMU_SPIKE_FLAG[1] = 0; IMU_SPIKE_FLAG[2] = 0; IMU_SPIKE_FLAG[3] = 0;
IMU_SPIKE_FLAG[4] = 0; IMU_SPIKE_FLAG[5] = 0;

        //Console.Write("count");

        if ((count > 0) && ((IMU_SPIKE_FLAG[0] + IMU_SPIKE_FLAG[1] + IMU_SPIKE_FLAG[2] + IMU_SPIKE_FLAG[3] +
IMU_SPIKE_FLAG[4] + IMU_SPIKE_FLAG[5]) == 0)) //Make sure the IMU data is new
        {

          //!!!!  IF YOU CHANGE ANYTHING HERE< YOU MUST UPDATE THE A-MATRIX   !!!!

          xm[0] = xK[0] + Ts * xK[3];  //Predict Position X
          xm[1] = xK[1] + Ts * xK[4];  //                 Y
          xm[2] = xK[2] + Ts * xK[5];  //                 Z


          //              Console.WriteLine("X " + xm[3].ToString() + "  X: " + xm[4].ToString() + "  Z: " + xm[5].ToString());
          //Predict Velocites (Vx,Vy,Vz)
          xm[3] = xK[3] + Ts * (C_Psi * C_Theta * u[0] + (-S_Psi * C_Phi + C_Psi * S_Theta * S_Phi) * u[1] + (S_Psi * S_Phi + C_Psi *
S_Theta * C_Phi) * (u[2]));//ok 19sep
          xm[4] = xK[4] + Ts * (S_Psi * C_Theta * u[0] + (C_Psi * C_Phi + S_Psi * S_Theta * S_Phi) * u[1] + (-C_Psi * S_Phi + S_Psi *
S_Theta * C_Phi) * (u[2]));//ok 19sep
          xm[5] = xK[5] + Ts * (G - S_Theta * u[0] + C_Theta * S_Phi * u[1] + C_Theta * C_Phi * u[2]);//ok 19sep2012


          //TRY THE LAWS OF PHYSICS CHEAT:
#if ZERO_VY_VZ
          xm[4] = 0;
          xm[5] = 0;
#endif


          xm[6] = xK[6] + Ts * (u[3] + u[4] * S_Phi * T_Theta + u[5] * C_Phi * T_Theta);
          xm[7] = xK[7] + Ts * (u[4] * C_Phi - u[5] * S_Phi);
          xm[8] = xK[8] + Ts * (u[4] * S_Phi + u[5] * C_Phi) * Sec_Theta;
```

```
            if(xm[8]<0) xm[8] += 2 * Math.PI;  //Not sure how this is going to work with the Jacobian, but hopefully it has small enough effect...
            //this was really buggering up at 0-360boundary


#if LOG_QR
            //Get noise in predictions (Q)
                                            //xp is just local variable to capture predictive part of xm so noise can be evaluated
            double[] xp = new double[9];
            //xp[0] = 0; //placeholders, variance is handled in velocity eqns
            //xp[1] = 0;
            //xp[2] = 0;
            xp[0] = xplast[0] + Ts * xK[3];
            xp[1] = xplast[1] + Ts * xK[4];
            xp[2] = xplast[2] + Ts * xK[5];

            //xp[3] = Ts * u[0];  //walking forward, at near level, instant velocity is average.3m/s (about right for wlaking)
            //xp[3] = xplast[3] + Ts * u[0];  //Runs away fast, increases every .1 second at average 0.3m/s?
            xp[3] = xplast[3]+Ts * (C_Psi * C_Theta * u[0] + (-S_Psi * C_Phi + C_Psi * S_Theta * S_Phi) * u[1] + (S_Psi * S_Phi + C_Psi *
S_Theta * C_Phi) * (u[2]));
            xp[4] = xplast[4] + Ts * (S_Psi * C_Theta * u[0] + (C_Psi * C_Phi + S_Psi * S_Theta * S_Phi) * u[1] + (-C_Psi * S_Phi + S_Psi *
S_Theta * C_Phi) * (u[2]));
            xp[5] = xplast[5] + Ts * (G - S_Theta * u[0] + C_Theta * S_Phi * u[1] + C_Theta * C_Phi * u[2]);

            //This gets a pure gyro attitude for reporting
            xp[6] = xplast[6] + Ts * (u[3] + u[4] * S_Phi * T_Theta + u[5] * C_Phi * T_Theta);
            xp[7] = xplast[7] + Ts * (u[4] * C_Phi - u[5] * S_Phi);
            xp[8] = xplast[8] + Ts * (u[4] * S_Phi + u[5] * C_Phi) * Sec_Theta;

            //This is just to get the gyro noise for tuning
            //xp[6] = Ts * (u[3] + u[4] * S_Phi * T_Theta + u[5] * C_Phi * T_Theta);
            //xp[7] = Ts * (u[4] * C_Phi - u[5] * S_Phi);
            //xp[8] = Ts * (u[4] * S_Phi + u[5] * C_Phi) * Sec_Theta;
            xplast = xp;

            //Prediction using previous EKF
//                String predictions = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9}, {8:F9}, {9:F9}",
DateTime.UtcNow.TimeOfDay.ToString(), xm[0], xm[1], xm[2], xm[3], xm[4], xm[5], xm[6], xm[7], xm[8]);
            //Just writes the sensor input integrations / true prediction
            String predictions = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9}, {8:F9}, {9:F9}",
DateTime.UtcNow.TimeOfDay.ToString(), xp[0], xp[1], xp[2], xp[3], xp[4], xp[5], xp[6], xp[7], xp[8]);


            Qlogfile.WriteLine(predictions);
//              Qlogfile.WriteLine(predictions);
            //Get noise in measurements (R)
            String measurements = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9}, {8:F9}, {9:F9}",
DateTime.UtcNow.TimeOfDay.ToString(), z[0], z[1], z[2], z[3], z[4], z[5], z[6], z[7], z[8]);
            Rlogfile.WriteLine(measurements);



#endif
            //BK17SEP2012:  Verified fx above as well as A matrix for roll,pitch,yaw.   Just need to add measurement
            //            of attitude to Kalman estimate step (right now only prediction for attitude)

            //              double TESTROLL = xm[6] * 180.0 / Math.PI;
            //              double TESTPITCH = xm[7] * 180.0 / Math.PI;
            //              Console.WriteLine("Roll: " + TESTROLL.ToString() + "  Pitch: " + TESTPITCH.ToString());



            //BK17SEP2012:  Z position row did not look correct, should be same as x,y.  made change
            //row 1  (x)
            A[0] = 1;
            A[9] = 0;
            A[18] = 0;
            A[27] = Ts;
            A[36] = 0;
            A[45] = 0;
```

```
	A[54] = 0;
	A[63] = 0;
	A[72] = 0;
	//Row 2 (y)
	A[1] = 0;
	A[10] = 1;
	A[19] = 0;
	A[28] = 0;
	A[37] = Ts;
	A[46] = 0;
	A[55] = 0;
	A[64] = 0;
	A[73] = 0;
	//Row 3  (z)
	A[2] = 0;
	A[11] = 0;
	A[20] = 1;
	A[29] = 0;
	A[38] = 0;
	A[47] = Ts;  //BK17SEP, Wrong partial derivative repaired: was not passing Z before nor taking discrete integral of Vz
	A[56] = 0;
	A[65] = 0;
	A[74] = 0;

	//BK17SEP2012:  Need to check the velocity rows below, they don't look quite right...
	//Row 4
	A[3] = 0;
	A[12] = 0;
	A[21] = 0;
	A[30] = 1;
	A[39] = 0;
	A[48] = 0;
	A[57] = Ts * ((S_Psi * S_Phi + C_Psi * S_Theta * C_Phi) * u[1] + (S_Psi * C_Phi - C_Psi * S_Theta * S_Phi) * (u[2]));//ok
19sep2012
	A[66] = Ts * (-C_Psi * S_Theta * u[0] + C_Psi * C_Theta * S_Phi * u[1] + C_Psi * C_Theta * C_Phi * (u[2]));//ok 19sep2012
	A[75] = Ts * (-S_Psi * C_Theta * u[0] - (C_Psi * C_Phi + S_Psi * S_Theta * S_Phi) * u[1] + (C_Psi * S_Phi - S_Psi * S_Theta *
C_Phi) * (u[2]));//ok19sep2012
	//Row 5
	A[4] = 0;
	A[13] = 0;
	A[22] = 0;
	A[31] = 0;
	A[40] = 1; //BK17SEP2012 Was using Vx instead of Vy
	A[49] = 0;
	A[58] = Ts * ((-C_Psi * S_Phi + S_Psi * S_Theta * C_Phi) * u[1] - (C_Psi * C_Phi + S_Psi * S_Theta * S_Phi) *
(u[2]));//ok19sep2012
	A[67] = Ts * (-S_Psi * S_Theta * u[0] + S_Psi * C_Theta * S_Phi * u[1] + S_Psi * C_Theta * C_Phi * (u[2]));//ok19sep2012
	A[76] = Ts * (C_Psi * C_Theta * u[0] + (-S_Psi * C_Phi + C_Psi * S_Theta * S_Phi) * u[1] + (S_Psi * S_Phi + C_Psi * S_Theta *
C_Phi) * (u[2]));//ok19sep2012

	//Row 6
	A[5] = 0;
	A[14] = 0;
	A[23] = 0;
	A[32] = 0;
	A[41] = 0;
	A[50] = 1;
	A[59] = Ts * (C_Theta * C_Phi * u[1] - C_Theta * S_Phi * (u[2]));//ok
	A[68] = -Ts * (C_Theta * u[0] + S_Theta * S_Phi * u[1] + S_Theta * C_Phi * (u[2]));//ok
	A[77] = 0;//ok

	//BK17SEP2012:  attitude rows below have all been vetted, they're ok.
	//Row 7
	A[6] = 0;
	A[15] = 0;
	A[24] = 0;
	A[33] = 0;
	A[42] = 0;
	A[51] = 0;
	A[60] = 1 + Ts * T_Theta * (u[4] * C_Phi - u[5] * S_Phi);
```

```
    A[69] = Ts * (Sec_Theta * Sec_Theta) * (u[4] * S_Phi + u[5] * C_Phi);
    A[78] = 0;
    //Row 8
    A[7] = 0;
    A[16] = 0;
    A[25] = 0;
    A[34] = 0;
    A[43] = 0;
    A[52] = 0;
    A[61] = Ts * -(u[4] * S_Phi + u[5] * C_Phi); //BK19SEP2012 was missing '-'
    A[70] = 1;
    A[79] = 0;
    //Row 9
    A[8] = 0;
    A[17] = 0;
    A[26] = 0;
    A[35] = 0;
    A[44] = 0;
    A[53] = 0;
    A[62] = Ts * Sec_Theta * (u[4] * C_Phi - u[5] * S_Phi);  //BK17SEP2012 Was missing sec
    A[71] = Ts * Sec_Theta * T_Theta * (u[4] * S_Phi + u[5] * C_Phi);
    A[80] = 1;


    //A Transpose
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
        {
            AT[j + i * 9] = A[i + j * 9];

        }
    }



    // A*P
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            for (temp[i + j * 9] = 0, k = 0; k < 9; k++)
                temp[i + j * 9] += (A[i + k * 9]) * (PK[k + j * 9]);

    //A*P*AT
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            for (temp1[i + j * 9] = 0, k = 0; k < 9; k++)
                temp1[i + j * 9] += (temp[i + k * 9]) * (AT[k + j * 9]);
    // (A*P*AT)+Q
    for (i = 0; i < 81; i++)
        Pm[i] = temp1[i] + Q[i];
}
else
{
    Console.WriteLine("IMU SPIKE");
    for (i = 0; i < 9; i++)
        xm[i] = xK[i];
    for (i = 0; i < 81; i++)
        Pm[i] = PK[i]; //changed here
}
//Not first step No GPS spike flag, and GPS is new
if (((SPIKE_FLAG[0] + SPIKE_FLAG[1] + SPIKE_FLAG[2] + SPIKE_FLAG[3] + SPIKE_FLAG[4] + SPIKE_FLAG[5]) == 0))
{
    //Pm*HT
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            for (temp2[i + j * 9] = 0, k = 0; k < 9; k++)
                temp2[i + j * 9] += (Pm[i + k * 9]) * (HT[k + j * 9]);

    //H*Pm
    for (i = 0; i < 9; i++)
```

```
                for (j = 0; j < 9; j++)
                    for (temp3[i + j * 9] = 0, k = 0; k < 9; k++)
                        temp3[i + j * 9] += (H[i + k * 9]) * (Pm[k + j * 9]);
            //H*P*H'
            for (i = 0; i < 9; i++)
                for (j = 0; j < 9; j++)
                    for (temp4[i + j * 9] = 0, k = 0; k < 9; k++)
                        temp4[i + j * 9] += (temp3[i + k * 9]) * (HT[k + j * 9]);

            // HPHT+R

            //BK17SEP2012 Need to figure out why extra space is needed, prob becasue of svd algorithm.
            for (i = 0; i < 84; i++)
                rwv[i] = 0.0;
            for (i = 0; i < 81; i++)
                rwv[i] = temp4[i] + R[i];
            rwv[180] = .0001;
            svdcmp2(rwv, 9, 9);

            //BK17SEP2012  What does this do?  I think this whole mess gives me the inverse(HPHT+R)
            t = 9 * rwv[9 * 9] * 2.220446049250313e-016 * rwv[(9 + 9 + 2) * 9];


            for (i = 0; i < 9; i++)
                for (j = 0; j < 9; j++)
                    for (temp6[i + j * 9] = 0, k = 0; k < 9; k++)
                        temp6[i + j * 9] += rwv[9 * (9 + 1) + i + k * 9] * ((rwv[9 * 9 + k]) > (t) ? (1 / rwv[9 * 9 + k]) : (0.0)) * rwv[j + k * 9];
            //K

            for (i = 0; i < 9; i++)
                for (j = 0; j < 9; j++)
                    for (K[i + j * 9] = 0, k = 0; k < 9; k++)
                        K[i + j * 9] += (temp2[i + k * 9]) * (temp6[k + j * 9]);


            //Calculate Residual Vector
            for (i = 0; i < 9; i++)
                Res[i] = z[i] - xm[i];

            //Fix for 0-360 boundary issue, when tuned and running properly no residuals over 1 rad let alone 6.   Make 180 breakpoint
            if (Math.Abs(Res[8]) > Math.PI) Res[8] -= 2 * Math.PI * Math.Sign(Res[8]); //get rid of 2pi swing

#if LOG_RES
            String residuals = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9}, {8:F9}, {9:F9}",
DateTime.UtcNow.TimeOfDay.ToString(), Res[0], Res[1], Res[2], Res[3], Res[4], Res[5], Res[6], Res[7], Res[8]);
            Reslogfile.WriteLine(residuals);
#endif


//Thisis where things get ugly.   If my simple understanding of covariance is right we should just be looking at the diagonal.
            //Gonna try it
            //K*Res
            for (i = 0; i < 9; i++)
                for (j = 0; j < 1; j++)
                    for (Kef[i + j * 9] = 0, k = 0; k < 9; k++)
                    {
                        Kef[i + j * 9] += (K[i + k * 9]) * (Res[k + j * 9]);
                    }
/*
            for (i = 0; i < 9; i++)
                    {
                        Kef[i] = (K[i*10]) * (Res[i]); //Diagonals are 0,10,20...
                    }
            */
#if LOG_K
            //Trying to log gain
            String gains = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9}, {8:F9}, {9:F9}",
DateTime.UtcNow.TimeOfDay.ToString(), K[0], K[10], K[20], K[30], K[40], K[50], K[60], K[70], K[80]);
            Klogfile.WriteLine(gains);
#endif
```

```
    //update the states
    for (i = 0; i < 9; i++)
        xUP[i] = xm[i] + Kef[i];

    // K*H
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            for (KH[i + j * 9] = 0, k = 0; k < 9; k++)
                KH[i + j * 9] += (K[i + k * 9]) * (H[k + j * 9]);



    //I-KH
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
        {
            if (i == j)
                IKH[i + j * 9] = 1 - KH[i + j * 9];
            else
                IKH[i + j * 9] = -KH[i + j * 9];
        }
    }

    //update the Error Covariance
    for (i = 0; i < 9; i++)
        for (j = 0; j < 9; j++)
            for (PUP[i + j * 9] = 0, k = 0; k < 9; k++)
                PUP[i + j * 9] += (IKH[i + k * 9]) * (Pm[k + j * 9]);
    //Make sure the angles stay within the ranges
    if (xUP[6] > 2 * PI)
        xUP[6] = xUP[6] - 2 * PI;
    else
    {
        if (xUP[6] < -2 * PI)
            xUP[6] = xUP[6] + 2 * PI;
    }
    if (xUP[7] > PI / 2)
        xUP[7] = xUP[7] - PI / 2;
    else
    {
        if (xUP[7] < -PI / 2)
            xUP[7] = xUP[7] + PI / 2;
    }
    if (xUP[8] > 2 * PI)
        xUP[8] = xUP[8] - 2 * PI;
    else
    {
        if (xUP[8] < 0)
            xUP[8] = xUP[8] + 2 * PI;
    }

    for (i = 0; i < 9; i++)
        x[i] = xUP[i];
    for (i = 0; i < 81; i++)
        p[i] = PUP[i];

}
else //if don't do measurement update
{
    Console.WriteLine("GPS SPIKE");
    if (xm[6] > 2 * PI)
        xm[6] = xm[6] - 2 * PI;
    else
    {
        if (xm[6] < -2 * PI)
            xm[6] = xm[6] + 2 * PI;
    }
    if (xm[7] > PI / 2)
```

```
                xm[7] = xm[7] - PI / 2;
            else
            {
                if (xm[7] < -PI / 2)
                    xm[7] = xm[7] + PI / 2;
            }
            if (xm[8] > 2 * PI)
                xm[8] = xm[8] - 2 * PI;
            else
            {
                if (xm[8] < 0)
                    xm[8] = xm[8] + 2 * PI;
            }
        }

        //BKAUG292012:  This short circuited the EKF!
        //              string s = "";
        //              for (i = 0; i < 9; i++) {
        //                  x[i] = xm[i];
        //                  s += x[i].ToString() + "    ";
        //              }
        //Console.WriteLine(s);
        //count++;



        //for (i = 0; i < 81; i++)
        // p[i] = Pm[i];



    }//DoIt

  }

}
```

A.1.1.4 Telemetry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO;


namespace mars.RCU.Telemetry
{


    public enum TelemetryDevices
    {
        IMU = 0,
        ARM = 1
    }


    public class TelemetryEventArgs<T> : EventArgs
    {
        public TelemetryEventArgs(T value)
        {
            m_value = value;
        }

        private T m_value;

        public T Value
        {
            get { return m_value; }
        }
    }



    public class TelemetryEngine
    {
        Models.IMU mars_imu;
        private StreamWriter logfile;
        string          Filename          =          string.Format("IMULOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv",          DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
        public TelemetryEngine()
        {
            // add mars imu
            mars_imu = new Models.IMU( "COM1", 115200, 500 );  //ask for imu telemetry events every 500 ms
            mars_imu.TelemetryUpdated_Event += new EventHandler<TelemetryEventArgs<object>>(mars_device_TelemetryUpdated_Event);

            //Open log file
            if (!File.Exists(Filename))
            {
                logfile = File.CreateText(Filename);
            }
        }


        /// <summary>
        /// Activates All Telemetry Devices
        /// </summary>
        public void Activate()
        {
            mars_imu.Activate_Telemetry();
        }


        /// <summary>
        /// Deactivates All Telemetry Devices
        /// </summary>
        public void Deactivate()
        {
            mars_imu.Deactivate_Telemetry();
            logfile.Close();
        }


        /// <summary>
        /// Event Handler for All Mars Telemetry Devices
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        void mars_device_TelemetryUpdated_Event(object sender, TelemetryEventArgs<object> e)
        {
            if (sender is Models.IMU )
            {
                //Models.IMU.IMU_Data data = (Models.IMU.IMU_Data)e.Value;


                //String     sIMUtelemetry     =     String.Format("{0:S},     {1:F9},     {2:F9},     {3:F9},     {4:F9},     {5:F9},     {6:F9},     {7:F9},
{8:F9},{9:F9},{10:F9},{11:F9},{12:F9},{13:F9},{14:F9}",              DateTime.UtcNow.TimeOfDay.ToString(),              data.gps_no_imu_lat,
data.gps_no_imu_lon,          data.gps_imu_lat,          data.gps_imu_lon,          data.gps_no_imu_velocity_X,          data.gps_imu_velocity_X,
data.gps_no_imu_velocity_Y, data.gps_imu_velocity_Y, data.gps_no_imu_velocity_Z, data.gps_imu_velocity_Z, data.imu_roll, data.imu_pitch,
data.imu_heading, data.imu_headingNOTILT);
                //Console.WriteLine(sIMUtelemetry);
                //logfile.WriteLine(sIMUtelemetry);

            }
        }
    }
}
```

A.1.1.5 IProvideTelemetry.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace mars.RCU.Telemetry
{
  interface IProvideTelemetry
  {

    // todo: add deactivated events()

    /// <summary>
    /// Activate Telemetry for the Device
    /// </summary>
    void Activate_Telemetry();

    /// <summary>
    /// Deactivate Telemetry for the Device
    /// </summary>
    void Deactivate_Telemetry();

    /// <summary>
    /// Returns whether the telemetry device is active
    /// </summary>
    /// <returns></returns>
    Boolean IsActive();

    /// <summary>
    /// Sets the rate at which the Telemetry available event will fire
    /// If it is set to zero, then telemetry will fire when it's available
    /// Defaults to 1 second
    /// </summary>
    /// <param name="RateInMilliSeconds"></param>
    void SetTelemetryRate(int RateInMilliSeconds);

    /// <summary>
    /// Returns the telemetry rate
    /// </summary>
    /// <returns></returns>
    int GetTelemetryRate();

    /// <summary>
    /// Event Handler for Updated Telemetry
    /// </summary>
    event EventHandler<TelemetryEventArgs<Object>> TelemetryUpdated_Event;
  }
}
```

A.1.1.6 ECEF2LLA.cs

```
/*
% ECEF2LLA - convert earth-centered earth-fixed (ECEF)
%          cartesian coordinates to latitude, longitude,
%          and altitude
%
% USAGE:
% [lat,lon,alt] = ecef2lla(x,y,z)
%
% lat = geodetic latitude (radians)
% lon = longitude (radians)
% alt = height above WGS84 ellipsoid (m)
% x = ECEF X-coordinate (m)
% y = ECEF Y-coordinate (m)
% z = ECEF Z-coordinate (m)
%
% Notes: (1) This function assumes the WGS84 model.
%       (2) Latitude is customary geodetic (not geocentric).
%       (3) Inputs may be scalars, vectors, or matrices of the same
%           size and shape.  Outputs will have that same size and shape.
%       (4) Tested but no warranty; use at your own risk.
%       (5) Michael Kleder, April 2006

function [lat,lon,alt] = ecef2lla(x,y,z)

% WGS84 ellipsoid constants:
a = 6378137;
e = 8.1819190842622e-2;

% calculations:
b   = sqrt(a^2*(1-e^2));
ep  = sqrt((a^2-b^2)/b^2);
p   = sqrt(x.^2+y.^2);
th  = atan2(a*z,b*p);
lon = atan2(y,x);
lat = atan2((z+ep^2.*b.*sin(th).^3),(p-e^2.*a.*cos(th).^3));
N   = a./sqrt(1-e^2.*sin(lat).^2);
alt = p./cos(lat)-N;

% return lon in range [0,2*pi)
lon = mod(lon,2*pi);

% correct for numerical instability in altitude near exact poles:
% (after this correction, error is about 2 millimeters, which is about
% the same as the numerical precision of the overall function)

k=abs(x)<1 & abs(y)<1;
alt(k) = abs(z(k))-b;

return
*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace mars.RCU.Models
{
    class ECEF2LLA
    {
        // WGS84 ellipsoid constants:
        private const Double a = 6378137;
        private const Double e = 8.1819190842622e-2;
        private const Double pi = 3.141592653589793;
        /// <summary>
        /// convert ecef x,y,z (meters) to lat/long in degrees
        /// </summary>
        /// <param name="x">x is input</param>
        /// <param name="y">y is input</param>
        /// <param name="z">z is input</param>
        /// <param name="Lat">Lat is output</param>
        /// <param name="Lon">Lon is output</param>
        public static void ECF2LLA(Double x, Double y, Double z, ref Double Lat, ref Double Lon)
        {
            //Console.WriteLine("x=" + x.ToString());
            //Console.WriteLine("y=" + y.ToString());
            //Console.WriteLine("z=" + z.ToString());
            Double b = Math.Sqrt(Math.Pow(a,2) * (1- Math.Pow(e,2)));

            Double ep  = Math.Sqrt((Math.Pow(a,2)-Math.Pow(b,2)) / Math.Pow(b,2));
            Double p   = Math.Sqrt(Math.Pow(x,2) + Math.Pow(y,2));
            Double th  = Math.Atan2(a*z, b*p);
            Lon = Math.Atan2(y,x);
            Lat = Math.Atan2((z + Math.Pow(ep,2) * b * Math.Pow(Math.Sin(th),3)), (p-Math.Pow(e,2) * a * Math.Pow(Math.Cos(th),3)));
            // Convert from Radians to Degrees
            Lon *= ( 180.0 / pi);
            Lat *= (180.0 / pi);

            //N   = a./sqrt(1-e^2.*sin(lat).^2);
            //alt = p./cos(lat)-N;
        }
    }
}
```

A.2 IMU_Player Project

The IMU_Player project reprocesses raw NAS data (collected using the IMU_Acq project). It runs faster than real-time to permit quick batch processing of recorded data for analysis and EKF tuning

A.2.1 Source Code List

A.2.1.1 EKF.cs

This file is the same as that used by IMU_Acq

A.2.1.2 IProvideTelemetry.cs

This file is the same as that used by IMU_Acq

A.2.1.3 ECEF2LLA.cs

This file is the same as that used by IMU_Acq

A.2.1.4 Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using mars.RCU.Telemetry;
using System.IO;

namespace IMU_Acq
{
    class Program
    {
        static void Main(string[] args)
        {

            //PASS FILENAME IN ARGS
            //CAN I SIMPLY DROP THE TELEMETRY AND INSTANTIATE IMU HERE WITH A NEW METHOD TO PASS IN LINE OF
DATA?
            StreamReader playbackFile;
            try { playbackFile = new StreamReader(args[0]); }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Thread.Sleep(2000);
                return;
            }
            mars.RCU.Models.IMU mars_imu;
            mars_imu = new mars.RCU.Models.IMU();  //set up generic, insertion 'IMU'

            //Read file, build packet and cal
            String imuPktString;
            while (playbackFile.Peek() >= 0)
            {
                imuPktString = playbackFile.ReadLine();
                string[] pktVals = imuPktString.Split(new char[] { ',' });

                double[] pktData = new double[16];
                if (pktVals.Length >= 16)
                {
                    for (int i = 0; i < 16; i++)
                        pktData[i] = Convert.ToDouble(pktVals[i]);
                    mars_imu.GenerateIMUData(pktData);
                }
            }





        }
    }
}
```

A.2.1.5 IMU.cs

```
//#define LOG_RAW_DATA

//#define LOG_RAW_DATA
#define LOG_NAS_DATA
//#define SUBTRACT_GRAV //Use Roll and Pitch to subtract gravity acceleration


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.IO;
using System.Threading;
using System.Collections.Concurrent;
using mars.RCU.Telemetry;


namespace mars.RCU.Models
{
    public class IMU : IProvideTelemetry
    {
#if LOG_MAG_DATA
        private StreamWriter logfile;
        string filename = string.Format("MAGLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
#endif
#if LOG_RAW_DATA
        private StreamWriter RawDatalogfile;
        string Rawfilename = string.Format("RAWLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);

#endif
#if LOG_NAS_DATA
        private StreamWriter NASlogfile;
        string NASfilename = string.Format("NASplayerLOG_{0:d}-{1:d}-{2:d}_{3:d}-{4:d}-{5:d}.csv", DateTime.UtcNow.Month,
DateTime.UtcNow.Day, DateTime.UtcNow.Year, DateTime.UtcNow.Hour, DateTime.UtcNow.Minute, DateTime.UtcNow.Second);
#endif
        #region IMU Data Types and Vars
        private const double MAG_DECLINATION = -9.09*Math.PI/180.0;  //Mag Declination from web in rads
        private const  Double G = 9.80665;


        private const Byte PACKET_FIRST_BYTE = 0xaa;    // packet first byte
        private const Byte PACKET_SECOND_BYTE = 68;     // packet second byte
        private const Byte PACKET_THIRD_BYTE = 18;      // packet third byte
        private const Byte IMU_PACKET_SIZE = 71;




        private SerialPort serial_port;
        private Thread FindIMUPacket_Thread;
        private Boolean NotFinished = true;
        private ConcurrentQueue<Byte> BufferQ;
        private Byte[] last_packet;
        private Byte[] packet;


        // rotation matrix for gps/imu
        private Double[] R1 = new Double[16];   /**4x4 Rotation matrix obtained using GPSLocation.m file**/
        private Double[] R2 = new Double[16];   /***4x4 inverse rotation matrix**/


//CONSTANTS
        //        private Double[] cal_const = new Double[10] { 3.33e-3, 3.33e-3, 3.33e-3, 0.025, 0.025, 0.025, 0, 0, 0, 0 };   /*calibration const only
for accleration and rate gyros*/
        //          private Double[] cal_const = new Double[10] { 3.33e-3, 3.33e-3, 3.33e-3, 0.0025, 0.0025, 0.0025, 0.5e-3, 0.5e-3, 0.5e-3, 0.14 };
/*calibration const only for accleration and rate gyros (gyros are per second, divded to .05s interval*/
        //BK15Aug2012:  According to data manula, cal const for gyro is 0.05 deg/s -> convert to rads 0.000873.   also added mag
        private Double[] cal_const = new Double[10] { 3.33e-3, 3.33e-3, 3.33e-3, 0.000873, 0.000873, 0.000873, 0.5e-3, 0.5e-3, 0.5e-3, 0.14 };
/*calibration const only for accleration and rate gyros*/


        private double Fil_x, Fil_y, Fil_z;
        private double Ecef_x, Ecef_y, Ecef_z;

        private double imu_mag_X, imu_mag_Y, imu_mag_Z; //raw acc values for attitude calculation
        private double imu_acc_X, imu_acc_Y, imu_acc_Z; //raw acc values for heading calculation
        private double magXoffset, magYoffset, magZoffset;

        private double gfield;                                //gravitational field at rover location for calculating attitude
        private double AccRoll, AccPitch;  //Accelerometer only pitch/roll for measurement input to EKF
        private double MagHeading;  //MAgnetometer heading with tilt comp for mesuremetn input to EKF
        double sinpitch, cospitch, sinroll, cosroll, Xh, Yh;              //vars to calculate tilt compensated heading

        private short[] uncal_imu_data = new short[10];    /*uncalibrated IMU data*/
        private double[] engr_imu_data = new Double[10];        /*calibrated IMU data*/
        private double[] gps_data = new Double[6] { 0, 0, 0, 0, 0, 0 };
        private Boolean is_active = false;
        private int EventRate = 1000;
        private DateTime LastHeardFromIMU;
        private DateTime TimeEventLastRaised;

        //public data
        public double gps_no_imu_lat;
        public double gps_no_imu_lon;
        public double gps_no_imu_velocity_X;
        public double gps_no_imu_velocity_Y;
        public double gps_no_imu_velocity_Z;
        public double gps_imu_lat;
        public double gps_imu_lon;
        public double gps_imu_velocity_X;
        public double gps_imu_velocity_Y;
        public double gps_imu_velocity_Z;
        public double imu_roll;
        public double imu_pitch;
        public double imu_heading;
        public double imu_headingNOTILT;
        #endregion

        #region Public Functions

        /// <summary>
        /// Event Raised for New IMU Telemetry Data
        /// </summary>
        public event EventHandler<TelemetryEventArgs<Object>> TelemetryUpdated_Event;
        public event EventHandler<TelemetryEventArgs<Object>> TelemetryUpdated;
        //public event EventHandler<Telemetry.TelemetryEventArgs> TelemetryUpdated;



        public IMU()
        {


//New TB data
            magXoffset = 0.1819;
            magYoffset = -0.2966;
            magZoffset = 0.0;
#if LOG_MAG_DATA
            if (!File.Exists(filename))
            {
                logfile = File.CreateText(filename);
            }
#endif
#if LOG_RAW_DATA
            if (!File.Exists(Rawfilename))
            {
                RawDatalogfile = File.CreateText(Rawfilename);
            }
#endif
#if LOG_NAS_DATA
            if (!File.Exists(NASfilename))
            {
                NASlogfile = File.CreateText(NASfilename);
                NASlogfile.WriteLine("Time,    gps_no_imu_lat,    gps_no_imu_lon,    gps_imu_lat,    gps_imu_lon,    gps_no_imu_velocity_X,
gps_imu_velocity_X,  gps_no_imu_velocity_Y,  gps_imu_velocity_Y,  gps_no_imu_velocity_Z,  gps_imu_velocity_Z,  imu_roll,  imu_pitch,
imu_heading, imu_headingNOTILT");
            }
#endif
            //1313 Montrose Backyard
            R1[0] = 0.111505018570193;
            R1[1] = -0.628023349132296;
            R1[2] = -0.770164465408730;
            R1[3] = -21003.6732229171;
            R1[4] = 0.984601274790237; //was wrong
            R1[5] = 0.174815130012935;
            R1[6] = 0;
            R1[7] = 0.00298616569489241;
            R1[8] = 0.134636401151770;
            R1[9] = -0.758304914439577;
            R1[10] = 0.637845354472136;
            R1[11] = -6369733.37545383;
            R1[12] = 0;
            R1[13] = 0;
            R1[14] = 0;
            R1[15] = 1;

            R2[0] = 0.111505018570193;
            R2[1] = 0.984601274790238;
            R2[2] = 0.134636401151770;
            R2[3] = 859939.990000000;
            R2[4] = -0.628023349132296;
            R2[5] = 0.174815130012935;
            R2[6] = -0.758304914439577;
            R2[7] = -4843390.92000000;
            R2[8] = -0.770164465408730;
            R2[9] = 0;
            R2[10] = 0.637845354472137;
            R2[11] = 4046728.56000000;
            R2[12] = 0;
            R2[13] = 0;
            R2[14] = 0;
            R2[15] = 1;


            //Setup EKF
            // init imu matrices
            EKF.init();
        }





        public IMU(String ComPort, int BaudRate, int TelemetryRateMilliSec)
        {


            // populate initial conditions - these are for morgantown.   when in houston, provide tanmay with X,Y,Z ECEF
            // and he will generate new initial conditions to plug in here.   see comments later.
            // todo, add this to config file.

            //Hard Coding Hard Iron offsets from MATLAB calibration
            //need to set this internally....  later
            magXoffset = 0.2648;
            magYoffset = -0.2430;
            magZoffset = -0.2127;
            //          magXoffset = 0.2580;
            //          magYoffset = -0.2510;
            //          magZoffset = -0.2054;

#if LOG_MAG_DATA
            if (!File.Exists(filename))
            {
               logfile = File.CreateText(filename);
            }
#endif
#if LOG_RAW_DATA
            if (!File.Exists(Rawfilename))
            {
               RawDatalogfile = File.CreateText(Rawfilename);
            }
#endif
#if LOG_NAS_DATA
            if (!File.Exists(NASfilename))
            {
               NASlogfile = File.CreateText(NASfilename);
               NASlogfile.WriteLine("Time, gps_no_imu_lat, gps_no_imu_lon, gps_imu_lat, gps_imu_lon, gps_no_imu_velocity_X,
gps_imu_velocity_X, gps_no_imu_velocity_Y, gps_imu_velocity_Y, gps_no_imu_velocity_Z, gps_imu_velocity_Z, imu_roll, imu_pitch,
imu_heading, imu_headingNOTILT");
            }
#endif


            //1313 Montrose Backyard
            R1[0] = 0.111504478325005;
            R1[1] = -0.628023172009515;
            R1[2] = -0.770164688058714;
            R1[3] = -21003.71124252398;
            R1[4] = 0.984601412089416; //was wrong
            R1[5] = 0.17814356708849;
            R1[6] = 0;
            R1[7] = 0.037836051546037;
            R1[8] = 0.134635844502855;
            R1[9] = -0.758305239404014;
            R1[10] = 0.637845085633984;
            R1[11] = -6369733.354108809;
            R1[12] = 0;
            R1[13] = 0;
            R1[14] = 0;
            R1[15] = 1;

            R2[0] = 0.111504478325005;
            R2[1] = 0.984601412089416;
            R2[2] = 0.134635844502855;
            R2[3] = 859936.4;
            R2[4] = -0.628023172009515;
            R2[5] = 0.174814356708849;
            R2[6] = -0.758305239404014;
            R2[7] = -4843393.000000002;
            R2[8] = -0.770164688058714;
            R2[9] = 0;
            R2[10] = 0.637845085633984;
            R2[11] = 4046726.80000000;
            R2[12] = 0;
            R2[13] = 0;
            R2[14] = 0;
            R2[15] = 1;

            // create the serial port
            serial_port = new SerialPort(ComPort, BaudRate);
            serial_port.RtsEnable = false;
            serial_port.DataBits = 8;
            serial_port.Parity = Parity.None;
            serial_port.StopBits = StopBits.One;
            serial_port.DataReceived += new SerialDataReceivedEventHandler(serial_port_DataReceived);
            serial_port.ErrorReceived += new SerialErrorReceivedEventHandler(serial_port_ErrorReceived);
            serial_port.ReceivedBytesThreshold = 75;    //maximize chances of getting a "good" 71 byte packet

            LastHeardFromIMU = DateTime.UtcNow;
            TimeEventLastRaised = DateTime.UtcNow;

            // set the update event time
            EventRate = TelemetryRateMilliSec;

            // create the find packet thread
            FindIMUPacket_Thread = new Thread(new ThreadStart(FindPacket_DoWork));
            FindIMUPacket_Thread.IsBackground = true;
        }

        /* todo - read imu config from file
         * public IMU(IMUConfiguration config)
        {

        }
         */

        /// <summary>
        /// Activate IMU Telemetry
        /// </summary>
        public void Activate_Telemetry()
        {
            if (!is_active)
            {
                // init imu matrices
                EKF.init();

                // generate the queue - also clears it
                BufferQ = new ConcurrentQueue<byte>();

                // close the serial port if it is open
                if (serial_port.IsOpen)
                {
                    serial_port.Close();
                }

                // start the thread
                NotFinished = true;
                switch (FindIMUPacket_Thread.ThreadState)
                {
                    case ThreadState.Background | ThreadState.Unstarted:
                        {
                            FindIMUPacket_Thread.Start();
                            break;
                        }
                    case ThreadState.Stopped:
                        {
                            // create the find packet thread
                            FindIMUPacket_Thread = new Thread(new ThreadStart(FindPacket_DoWork));
                            FindIMUPacket_Thread.IsBackground = true;
                            FindIMUPacket_Thread.Start();
                            break;
                        }
                }


                // open the serial port & clear buffers
                try
                {
                    serial_port.Open();
                    if (serial_port.IsOpen)
                    {
                        serial_port.DiscardInBuffer();
                        serial_port.DiscardOutBuffer();
                    }

                    // set that we are active
                    is_active = true;
                }
                catch (Exception ex)
                {
                    //Handle Exception
                }

            }
            else
            {
                //may not want to throw exception here?
                throw new Exception("Telemetry is already active! Cannot activate");
            }
        }


        /// <summary>
        /// Deactivate IMU Telemetry
        /// </summary>
        public void Deactivate_Telemetry()
        {
            if (is_active)
            {
                is_active = false;

                // close the serial port
                if (serial_port.IsOpen)
                {
                    serial_port.Close();
                }

                // shutdown the thread
                NotFinished = false;
                if (FindIMUPacket_Thread.ThreadState == ThreadState.Running)
                {
                    FindIMUPacket_Thread.Join();
                }
            }
            else
            {
```

```
                // may not want to throw exception here?
                throw new Exception("Telemetry is NOT active!  Cannot Deactivate");
            }
        }

        /// <summary>
        /// Returns Whether IMU Telemetry is Active
        /// </summary>
        public Boolean IsActive()
        {
            return is_active;
        }


        /// <summary>
        /// Sets the rate at which the Telemetry available event will fire
        /// If it is set to zero, then telemetry will fire when it's available
        /// Defaults to 1 second
        /// </summary>
        /// <param name="RateInMilliSeconds"></param>
        public void SetTelemetryRate(int RateInMilliSeconds)
        {
            EventRate = RateInMilliSeconds;
        }


        /// <summary>
        /// Returns the telemetry rate
        /// </summary>
        /// <returns></returns>
        public int GetTelemetryRate()
        {
            return EventRate;
        }

        #endregion

        #region Private Functions

        private void serial_port_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {

#if DEBUG_MSGS
//TimeSpan ts = DateTime.UtcNow.Subtract(LastHeardFromIMU);
//Console.WriteLine("TS = :{0:d}", ts.Milliseconds);
#endif
            //record that we heard from IMU - this is used in worker thread to ensure that thread is operating on fresh data
            LastHeardFromIMU = DateTime.UtcNow;


            SerialPort sp = (SerialPort)sender;
            int BytesAvail = sp.BytesToRead;    //copy to var asap in case it were to change

            byte[] data = new byte[BytesAvail];
            sp.Read(data, 0, BytesAvail);

            // add the serial port data to the queue
            try
            {
                // enqueue the serial port bytes
                lock (BufferQ)
                {
                    for (int i = 0; i < BytesAvail; i++)
                    {
                        BufferQ.Enqueue(data[i]);
                    }

                    // empty the queue if it's too big.   just blow it away - we dont want to backup
                    if (BufferQ.Count > (IMU_PACKET_SIZE * 10))   // 100 packets = 5 seconds of 20Hz telemetry as the cutoff chosen arbitrarily
                    {
```

```
                // clear a concurrent queue by allocating a new one.   This is the fastest way.   We could also dequeue 71 bytes but since
                // we dont care about losing data, i say we just dump it and grab the next fresh telemetry packet.

                Console.WriteLine("Clearing Queue - Count @  " + BufferQ.Count.ToString());
                BufferQ = new ConcurrentQueue<byte>();

                //Console.WriteLine("INFO: IMU Queue Too Big!  Not clearing!");
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception: " + ex.Message + " in " + ex.TargetSite.ToString());
    }
}

private void serial_port_ErrorReceived(object sender, SerialErrorReceivedEventArgs e)
{

}

private void FindPacket_DoWork()
{
    Byte b1;
    Byte b2;
    Byte b3;
    Boolean ErrorOccurred = false;

    while (NotFinished)
    {
        ErrorOccurred = false;
        Thread.Sleep(10);

        //make sure we heard from IMU via serial port within 10 seconds.   If not, clear the Queue.
        //this ensures that if the IMU stops sending or the RCU stops getting data, stale packets in the
        //Q will not be sent and misconstrued for being "fresh".
        lock (BufferQ)
        {
            TimeSpan ts = DateTime.UtcNow.Subtract(LastHeardFromIMU);
            if (ts.TotalSeconds >= 10)
            {
                BufferQ = new ConcurrentQueue<byte>();
                Console.WriteLine("WARNING: IMU on Serial Port Not Heard From in 10 Seconds.   Dumping Stale Data");
            }

            // only process Q if it contains at least one packet
            if (BufferQ.Count > IMU_PACKET_SIZE)
            {
                try
                {
                    if (BufferQ.TryDequeue(out b1))
                    {
                        if (b1 == PACKET_FIRST_BYTE)
                        {
                            if (BufferQ.TryDequeue(out b2))
                            {
                                if (b2 == PACKET_SECOND_BYTE)
                                {
                                    if (BufferQ.TryDequeue(out b3))
                                    {
                                        if (b3 == PACKET_THIRD_BYTE)
                                        {
                                            // always allocate new memory for packet - do not reuse old
                                            packet = new byte[IMU_PACKET_SIZE];

                                            // packet found - put all bytes into a packet array
                                            packet[0] = b1;
                                            packet[1] = b2;
                                            packet[2] = b3;
                                            //  grab the rest of the bytes to form a packet
                                            for (int i = 3; i < (IMU_PACKET_SIZE - 3); i++)
                                            {
                                                // form the packet
                                                if (BufferQ.TryDequeue(out packet[i]))
                                                {
                                                    //tweak this as necessary

                                                }
                                                else
                                                {
                                                    Console.WriteLine("WARNING: Packet Dequeue Error.   Dumping Packet");
                                                    ErrorOccurred = true;
                                                }
                                            }

                                            //generate imu data and raise event
                                            if (!ErrorOccurred)
                                            {
                                                GenerateIMUData(packet);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Exception: " + ex.Message + " in " + ex.TargetSite.ToString());
                }
            }
        }
    }
}


/// <summary>
/// Generate the IMU data from the packet and raise the event with the deep-copy data
/// </summary>
/// <param name="packet"></param>

public void GenerateIMUData(double[] nas_data)
{

        // calibrated imu data?  SZ
        //bumped to 9 to get mag data
        for (int i = 0; i < 9; i++)
        {
            if (i < 3)
            {
                //engr_imu_data[i] = nas_data[i] * cal_const[i] * 9.8;      /**constructing engineering data for only acceleration and rate gyros,
magnetometer excluded*/
                engr_imu_data[i] = nas_data[i] * cal_const[i] * G;    //constructing engineering data for only accelerometers, trying -G
            }
            else
            {
                // removed 9.8
                engr_imu_data[i] = nas_data[i] * cal_const[i];    /**constructing engineering data for rate gyros and magnetometers*/
            }
        }

        //Convention:   EAST-NORTH-DOWN (all positive), nose up = pitch positive, right-side down = roll positive, turn clockwise =
heading/yaw positive
        //Flip signs of gyro Gy, Gz   Ax  (to match END and std attitude conventions)  and Mz (to match Honeywell equations
        engr_imu_data[4] *= -1;  //Flip Gy
        engr_imu_data[5] *= -1;  //Flip Gz
        engr_imu_data[0] *= -1;  //Flip Ax
```

```
            //Try Flipping My and Mz anbd using Freescale AN to calculate heading
            engr_imu_data[7] *= -1;  //Flip My
            engr_imu_data[8] *= -1;  //Flip Mz




            //copy raw IMU values to local vars that make more sense for tracking through eqns
            //imu_acc_X = engr_imu_data[0];
            imu_acc_X = engr_imu_data[0];
            imu_acc_Y = engr_imu_data[1];
            imu_acc_Z = engr_imu_data[2];
            imu_mag_X = engr_imu_data[6];
            imu_mag_Y = engr_imu_data[7];
            imu_mag_Z = engr_imu_data[8];

            gfield = 9.80665;  //use same constant as the EKF does.

            //Both Freescal AN3461 and Kim have it this way, I think I buggered itup!
            AccPitch = Math.Asin(imu_acc_X / gfield);
            AccRoll = Math.Asin(-imu_acc_Y/(gfield*Math.Cos(AccPitch)));
            Console.WriteLine("A_ROLL: {0:f2}   A_PITCH:  {1:f2}", AccRoll * 180.0 / Math.PI, AccPitch * 180.0 / Math.PI);

            //Put x and y mag field on horizontal plane to tilt compensate, then calculate heading
            //min calc time by calling trig functions once

            //Use last EKF output for tilt compensation

            sinpitch = Math.Sin(EKF.x[7]);
            sinroll = Math.Sin(EKF.x[6]);
            cospitch = Math.Cos(EKF.x[7]);
            cosroll = Math.Cos(EKF.x[6]);


        //Take Hard Iron out
            imu_mag_X -= magXoffset;
            imu_mag_Y -= magYoffset;
            imu_mag_Z -= magZoffset;

            Xh = imu_mag_X * cospitch + imu_mag_Y * sinpitch*sinroll + imu_mag_Z * sinpitch * cosroll;
            Yh = imu_mag_Z * sinroll - imu_mag_Y * cosroll;




            MagHeading = Math.Atan2(Yh, Xh) + MAG_DECLINATION;
            //After fighting with this and realizing there is no way to calibrate z axis on rover, trying un tilt compensated
            MagHeading = Math.Atan2(-imu_mag_Y, imu_mag_X) + MAG_DECLINATION;
           if (MagHeading < 0) MagHeading += 2 * Math.PI;  //MUST CORRECT PREDICTION TOO in EKF!!




            imu_headingNOTILT = Math.Atan2(-imu_mag_Y, imu_mag_X) + MAG_DECLINATION;
            if (imu_headingNOTILT < 0) imu_headingNOTILT += 2*Math.PI;
            imu_headingNOTILT = imu_headingNOTILT * 180.0 / Math.PI;  //Convert to degrees





            Console.WriteLine("MX: {0:F3}  MY: {1:F3}  MZ: {2:F3}  XH: {3:F3}  YH: {4:F3}", imu_mag_X, imu_mag_Y, imu_mag_Z, Xh,
Yh);
            //Console.WriteLine("MX: {0:F3}  MY: {1:F3}  MZ: {2:F3}  XH: {3:F3}  YH: {4:F3}", imu_mag_X, imu_mag_Y, imu_mag_Z, Xh,
Yh);

#if LOG_MAG_DATA
            string sMAGreadings = string.Format("{0:S},{1:F3},{2:F3}", DateTime.UtcNow.TimeOfDay.ToString(), Xh, Yh);
            //string   sMAGreadings   =   string.Format("{0:S},{1:F3},{2:F3},{3:F3}",   DateTime.UtcNow.TimeOfDay.ToString(),   imu_mag_X,
imu_mag_Y, imu_mag_Z);
```

```
            //Console.WriteLine(sMAGreadings);
            logfile.WriteLine(sMAGreadings);
#endif


            /***constructing data for GPS, the default GPS data is in ECEF coordinates**/
            // note when in houston, send Tanmay the x,y,and z coord and he will provide a new R matrix that gets set in the
            // IMU constructor
            gps_data[0] = nas_data[10];    //x
            gps_data[1] = nas_data[11];    //y
            gps_data[2] = nas_data[12];    //z
            gps_data[3] = nas_data[13];    //vx
            gps_data[4] = nas_data[14];    //vy
            gps_data[5] = nas_data[15];    //vz



            //Write Raw values to CSV File for post processing and EKF tuning


            // fill the packet with raw gps data, no imu
            ECEF2LLA.ECF2LLA(gps_data[0], gps_data[1], gps_data[2], ref gps_no_imu_lat, ref gps_no_imu_lon);

#if LOG_RAW_DATA
            //write imu data
            for (int i = 0; i < 10; i++)
            {
               RawDatalogfile.Write(uncal_imu_data[i].ToString());
               RawDatalogfile.Write(",");
            }
            //write GPS data
            for (int i = 0; i < 6; i++)
            {
               RawDatalogfile.Write(gps_data[i].ToString());
               RawDatalogfile.Write(",");
            }
            RawDatalogfile.Write(gps_no_imu_lat.ToString());
            RawDatalogfile.Write(",");
            RawDatalogfile.Write(gps_no_imu_lon.ToString());

            RawDatalogfile.Write("\n");
#endif
            /***converting GPS data from ECEF to Local coordinate for which we need Rotation matrix (4X4) which can be generated using
GPSLocation.m file, R[0]=Rotation matrix(1,1),
            *R[1]=Rotation matrix(1,2),R[2]=Rotation matrix(1,3),R[3]=Rotation matrix(1,4),R[4]=Rotation matrix(2,1) and so on, local
coordinate=R*x and local velocity=R*v ***********************/

            //USe rotation matrix to convert raw GPS
            //init to zero, change only if GPS has value
            Double X_local = 0.0;
            Double Y_local = 0.0;
            Double Z_local = 0.0;
            if (gps_data[0] + gps_data[1] + gps_data[2] != 0.0)
            {
               X_local = R1[0] * gps_data[0] + R1[1] * gps_data[1] + R1[2] * gps_data[2] + R1[3];    // SZ: what is X?  Actual value or method to
calculate in C.   Not matlab.
               Y_local = R1[4] * gps_data[0] + R1[5] * gps_data[1] + R1[6] * gps_data[2] + R1[7];    // SZ: what is y?  Actual value or method to
calculate in C.   Not matlab.
               Z_local = R1[8] * gps_data[0] + R1[9] * gps_data[1] + R1[10] * gps_data[2] + R1[11];    // SZ: what is Z?  Actual value for method
to calculate in C.  Not matlab.
            }


            //BK28AUG:  CONVERSION IS OK

            Double Vx_local = R1[0] * gps_data[3] + R1[1] * gps_data[4] + R1[2] * gps_data[5];
            Double Vy_local = R1[4] * gps_data[3] + R1[5] * gps_data[4] + R1[6] * gps_data[5];
            Double Vz_local = R1[8] * gps_data[3] + R1[9] * gps_data[4] + R1[10] * gps_data[5];

            gps_no_imu_velocity_X = Vx_local;
```

```
                gps_no_imu_velocity_Y = Vy_local;
                gps_no_imu_velocity_Z = Vz_local;

                Double X_SWD = X_local;
                Double Y_SWD = Y_local;
                Double Z_SWD = Z_local;

                Double Vx_SWD = Vx_local;
                Double Vy_SWD = Vy_local;
                Double Vz_SWD = Vz_local;
                //BK17SEP2012:  combining gps measurement with measured attitude from accel/mag.   May want to look at better storage of this
                double[] measurement = new double[] { X_SWD, Y_SWD, Z_SWD, Vx_SWD, Vy_SWD, Vz_SWD, AccRoll, AccPitch,
MagHeading };

                EKF.ApplyFilter(engr_imu_data, measurement);

                Fil_x = EKF.x[0];
                Fil_y = EKF.x[1];
                Fil_z = EKF.x[2];

                Ecef_x = R2[0] * Fil_x + R2[1] * Fil_y + R2[2] * Fil_z + R2[3];
                Ecef_y = R2[4] * Fil_x + R2[5] * Fil_y + R2[6] * Fil_z + R2[7];
                Ecef_z = R2[8] * Fil_x + R2[9] * Fil_y + R2[10] * Fil_z + R2[11];


                ECEF2LLA.ECF2LLA(Ecef_x, Ecef_y, Ecef_z, ref gps_imu_lat, ref gps_imu_lon);
                gps_imu_velocity_X = EKF.x[3];  //Local coordinates
                gps_imu_velocity_Y = EKF.x[4];
                gps_imu_velocity_Z = EKF.x[5];

                //21AUG2012BK:  Is Attitude stored in 6.7.8?
                imu_roll = EKF.x[6] * 180.0 / Math.PI;
                imu_pitch = EKF.x[7] * 180.0 / Math.PI;
                imu_heading = EKF.x[8] * 180.0 / Math.PI;



                String sIMUtelemetry = String.Format("{0:S}, {1:F9}, {2:F9}, {3:F9}, {4:F9}, {5:F9}, {6:F9}, {7:F9},
{8:F9},{9:F9},{10:F9},{11:F9},{12:F9},{13:F9},{14:F9}", DateTime.UtcNow.TimeOfDay.ToString(), gps_no_imu_lat, gps_no_imu_lon,
gps_imu_lat, gps_imu_lon, gps_no_imu_velocity_X, gps_imu_velocity_X, gps_no_imu_velocity_Y, gps_imu_velocity_Y,
gps_no_imu_velocity_Z, gps_imu_velocity_Z, imu_roll, imu_pitch, imu_heading, imu_headingNOTILT);
                //Console.WriteLine("K_ROLL: {0:f2}   K_PITCH:  {1:f2}", imu_roll, imu_pitch);
                Console.WriteLine("K_HDG: {0:f2}", imu_heading);
                NASlogfile.WriteLine(sIMUtelemetry);


            }

        }





}
```

Appendix B. Analysis Scripts

The analysis scripts were written in MATLAB and process data recorded by NAS and Rover software for the purposes of analyzing data, testing software performance, and ultimately improving the algorithms implemented in the code.

B.1 MagCalibration Analysis Script

This MATLAB script processes recorded magnetometer data (from IMU_Acq and/or the RCU project) and provides hard-iron offsets. It also contains code to perform soft iron calibration, but due to the absence of soft-iron effects in testing, this has been commented out and not thoroughly tested.

B.1.1 MagCalibration.m

```
load magdata;

%Let's try a sample average with window size 10
i=1;
sum=[0 0 0];
aIdx = 1;
for idx=1:length(magdata)
    sum(1) = sum(1) + magdata(idx,1); %writing it in similar fashion to c implementation for porting later
    sum(2) = sum(2) + magdata(idx,2);
    sum(3) = sum(3) + magdata(idx,3);

    if(mod(idx,10) == 0)
       avgmagdata(aIdx,1)=sum(1)/10;
       avgmagdata(aIdx,2)=sum(2)/10;
       avgmagdata(aIdx,3)=sum(3)/10;
       sum=[0 0 0];
       aIdx= aIdx+1;
    end
end
% gamma = sum(3)/length(magdata); %just average z readings and subtract
magdata = avgmagdata;


% scatter(magdata(:,1),magdata(:,2),'b'); %make plot to show compensation
scatter3(magdata(:,1),magdata(:,2),magdata(:,3),'b'); %make plot to show compensation
% Already tilt compensated

%Outliers breaking SI and probably making HI inaccurate
%put a std deviation [2 sigma?] filter on data and see if can capture ellipse
%parameters then?

%Can't take SD or mean here, data is around the circle


%HARD IRON COMPENSATE
%alpha is x offset, beta is y offset
alpha = (max(magdata(:,1))+min(magdata(:,1)))/2;
beta = (max(magdata(:,2))+min(magdata(:,2)))/2;
gamma = (max(magdata(:,3))+min(magdata(:,3)))/2;

% Before Soft Iron compensation take offset off of all points
HImagdata = [magdata(:,1)-alpha,magdata(:,2)-beta,magdata(:,3)-gamma];
hold on;
% scatter(HImagdata(:,1),HImagdata(:,2),'r'); %show HI on plot
```

```
scatter3(HImagdata(:,1),HImagdata(:,2),HImagdata(:,3),'r'); %show HI on plot



% %SOFT IRON COMPENSATE
% %Perform SI on the tilt and HI corrected data
% r = sqrt(HImagdata(:,1).^2+HImagdata(:,2).^2); %Find the distances for all point combos
% %MAx will be the major axis and will determine theta of the 'twist' of
% %ellipse
% [maxr maxPairIdx] = max(r); %find major axis r
% [minr minPairIdx] = min(r); %fin minor axis q
% plot([0 HImagdata(maxPairIdx,1)],[0 HImagdata(maxPairIdx,2)],'k');
% plot([0 HImagdata(minPairIdx,1)],[0 HImagdata(minPairIdx,2)],'k');
%
% theta = asin(HImagdata(maxPairIdx,2)/maxr);
%
% %Create rotation matrix
% Rmat = [cos(theta) sin(theta); -sin(theta) cos(theta)];
%
% %Rotate ellipse
% ellipse = (Rmat * HImagdata')';
% scatter(ellipse(:,1),ellipse(:,2),'g'); %show HI on plot
%
% %Scale ellipse back to circle
%
%
% scale = minr/maxr;
%
% scaledellipse = [ellipse(:,1)./scale, ellipse(:,2)];
% scatter(scaledellipse(:,1),scaledellipse(:,2),'c'); %show HI on plot
%
% %Rotate back
% Rmat = [cos(-theta) sin(-theta); -sin(-theta) cos(-theta)];
% ellipse = (Rmat * scaledellipse')';
% scatter(ellipse(:,1),ellipse(:,2),'m'); %show HI on plot
```

B.2 Plot4Static Analysis Script

This MATLAB Script plots the 4 static positions surveyed by the NAS in respect to 2 meter radius error circles

B.2.1 Plot4Static.m

```
close all;  %will close figures open until now
clear;  %clear vars


%CONSTANTS
%distance from http://www.csgnetwork.com/degreelenllavcalc.html
LATMETERS = 111027.50; %METERS PER DGREE LAT
LONMETERS= 85852.93;%METERS PER DGREE LON
LATCORRECTION = 11;%meters, + move MAP north - move MAP south
LONCORRECTION = 0;%meters, + move MAP east - move MAP west

%Calcutlate dgree offset to adjust map
latoffset = LATCORRECTION/LATMETERS;
lonoffset = LONCORRECTION/LONMETERS;


% immap = imread('./maps/backyard.jpg');
immap = imread('./maps/ColPrkLotClose.jpg');
```

```
% for i=1:2;  %make 2 figures  (For this one only scatteer
    figure; image(immap);hold on;
    % coords = load('./maps/backyard.txt');  %Read coordinate vector in
    % [n,s,e,w] format
    coords = load('./maps/ColPrkLotClose.txt');  %Read coordinate vector in [n,s,e,w] format
    n=coords(1)+latoffset;
    s=coords(2)+latoffset;
    e=coords(3)+lonoffset;
    w=coords(4)+lonoffset;
    latpts = (n-s)/size(immap,1);
    lonpts = (e-w)/size(immap,2);
    yticks = n - latpts .* get(gca,'YTick');
    latTickLabs = cell(size(yticks));
    for yidx = 1:length(yticks)
        latTickLabs{yidx} = num2str(yticks(yidx),'% 2.4f');
    end


    xticks = w + lonpts * get(gca,'XTick');
    lonTickLabs = cell(size(xticks));
    for xidx = 1:length(xticks)
        lonTickLabs{xidx} = num2str(xticks(xidx),'% 2.4f');
    end


%     lonTickLabs = num2str(w + lonpts * get(gca,'XTick'),'% 2.5f');
%     latTickLabs = strrep(latTickLabs,'      ','|');
%     lonTickLabs = strrep(lonTickLabs,'     ','|');
    set(gca,'YTickLabel',latTickLabs);
    set(gca,'XTickLabel',lonTickLabs);

% end
%convert ticks to lat,lon in degrees


%Load the data from the rover NAS

%Read 4 static test points from NAS
tmpdata = xlsread('.\Data 13APR\NAS_STATIC_1\NASplayerLOG_4-14-2013_20-2-43_S1.csv');
data(:,:,1) = tmpdata(1:3000,:);  %only take first 5 minutes
tmpdata = xlsread('.\Data 13APR\NAS_STATIC_2\NASplayerLOG_4-14-2013_20-4-13_S2.csv');
data(:,:,2) = tmpdata(1:3000,:);  %only take first 5 minutes
tmpdata = xlsread('.\Data 13APR\NAS_STATIC_3\NASplayerLOG_4-14-2013_20-4-49_S3.csv');
data(:,:,3) = tmpdata(1:3000,:);  %only take first 5 minutes
tmpdata = xlsread('.\Data 13APR\NAS_STATIC_4\NASplayerLOG_4-14-2013_20-5-29_S4.csv');
data(:,:,4) = tmpdata(1:3000,:);  %only take first 5 minutes

%Iterate through all 4 data points
for dIdx =1:4
    gpslat(:,dIdx) = data(:,2,dIdx);
    gpslon(:,dIdx) = data(:,3,dIdx);
    ekflat(:,dIdx) = data(:,4,dIdx);
    ekflon(:,dIdx) = data(:,5,dIdx);

    %RAW GPS (Old Format, must update if ever use
     xgps(:,dIdx) = abs(w-gpslon(:,dIdx))/lonpts;
     ygps(:,dIdx) = (n-gpslat(:,dIdx))/latpts;


    xekf(:,dIdx) = abs(w-ekflon(:,dIdx))/lonpts;
    yekf(:,dIdx) = (n-ekflat(:,dIdx))/latpts;
    %  figure(1);scatter(xekf(1),yekf(1),'g')
    %  figure(1);scatter(xekf(end),yekf(end),'r')
end
%      figure(1);scatter(xgps,ygps,'b')
%      figure(2);plot(xgps,ygps,'b')
pClr=['b' 'g' 'r' 'c'];
for pIdx =1:4
```

```
    figure(1);scatter(xekf(:,pIdx),yekf(:,pIdx),[pClr(pIdx) 'x'])
%     figure(2);plot(xekf(:,pIdx),yekf(:,pIdx),pClr(pIdx))
end



%Use Hemi data to make error circles
%Read the Hemisphere Data
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_1_SURVEY.csv');
hemiData(:,:,1) = hd(1:1200,:);    %V100 does not really collect at 10Hz, more like 5Hz.
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_2_SURVEY.csv');
hemiData(:,:,2) = hd(1:1200,:);
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_3_SURVEY.csv');
hemiData(:,:,3) = hd(1:1200,:);
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_4_SURVEY.csv');
hemiData(:,:,4) = hd(1:1200,:);

for hIdx = 1:4
   hemiLat = hemiData(:,1,hIdx);
   hemiLat(isnan(hemiLat))=n; %shouldn't bias much
   hemiLon = hemiData(:,2,hIdx);
   hemiLon(isnan(hemiLon))=w; %shouldn't bias much
   %Use average to find center of error circle
   lonErrCtr = abs(w-mean(hemiLon))/lonpts;
   latErrCtr = (n-mean(hemiLat))/latpts;

   %For calculating error
   avgHemiLon(hIdx) = mean(hemiLon);
   avgHemiLat(hIdx) = mean(hemiLat);
%Circle plot code from http://www.mathworks.com/matlabcentral/answers/3058
   r = 2; %2 meter error radius
   ang=0:0.01:2*pi;
   xp=r*cos(ang);
   yp=r*sin(ang);
   %Convert to degrees->pixels
   xp = xp/LONMETERS/lonpts; %lon
   yp = yp/LATMETERS/latpts;%lat
   figure(1); plot(lonErrCtr+xp,latErrCtr+yp,pClr(hIdx),'LineWidth',1.5);

end
title('Static Error Plot with 2m Error Radius');grid on

legend(['Corner 1'; 'Corner 2' ;'Corner 3'; 'Corner 4']);
xlabel('Longitude');
ylabel('Latitude');




%Error in meters
for eIdx = 1:4
   ekflonerr = mean(abs(ekflon(:,eIdx)-avgHemiLon(eIdx))) * LONMETERS;
   ekflaterr = mean(abs(ekflat(:,eIdx)-avgHemiLat(eIdx))) * LATMETERS;
   errekf(eIdx) = sqrt(ekflaterr^2 + ekflonerr^2)
   gpslonerr = mean(abs(gpslon(:,eIdx)-avgHemiLon(eIdx))) * LONMETERS;
   gpslaterr = mean(abs(gpslat(:,eIdx)-avgHemiLat(eIdx))) * LATMETERS;
   gpserr(eIdx) = sqrt(gpslaterr^2 + gpslonerr^2)

end
```

B.3 Plot5Pass Analysis Script

This MATLAB script plots 5 passes of NAS data versus one pass of V100 data for comparison

B.3.1 Plot5Pass.m

```
close all;  %will close figures open until now
```

```
clear;  %clear vars


%CONSTANTS
%distance from http://www.csgnetwork.com/degreelenllavcalc.html
LATMETERS = 111027.86; %METERS PER DGREE LAT
LONMETERS= 85829.94;%METERS PER DGREE LON
LATCORRECTION = 11;%meters, + move MAP north - move MAP south
LONCORRECTION = 0;%meters, + move MAP east - move MAP west

%Calcutlate dgree offset to adjust map
latoffset = LATCORRECTION/LATMETERS;
lonoffset = LONCORRECTION/LONMETERS;





% immap = imread('./maps/backyard.jpg');
immap = imread('./maps/ColPrkLotClose.jpg');



for i=1:2;  %make 2 figures
    figure; image(immap);hold on;
    % coords = load('./maps/backyard.txt');  %Read coordinate vector in
    % [n,s,e,w] format
    coords = load('./maps/ColPrkLotClose.txt');  %Read coordinate vector in [n,s,e,w] format
    n=coords(1)+latoffset;
    s=coords(2)+latoffset;
    e=coords(3)+lonoffset;
    w=coords(4)+lonoffset;
    latpts = (n-s)/size(immap,1);
    lonpts = (e-w)/size(immap,2);
    yticks = n - latpts .* get(gca,'YTick');
    latTickLabs = cell(size(yticks));
    for yidx = 1:length(yticks)
        latTickLabs{yidx} = num2str(yticks(yidx),'% 2.4f');
    end


    xticks = w + lonpts * get(gca,'XTick');
    lonTickLabs = cell(size(xticks));
    for xidx = 1:length(xticks)
        lonTickLabs{xidx} = num2str(xticks(xidx),'% 2.4f');
    end

    set(gca,'YTickLabel',latTickLabs);
    set(gca,'XTickLabel',lonTickLabs);

end
%convert ticks to lat,lon in degrees


%Load the data from the rover NAS

%Read 5 motion test from NAS
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_1\NASplayerLOG_4-14-2013_20-6-20_M1.csv');
data(:,:,1) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_2\NASplayerLOG_4-14-2013_20-6-47_M2.csv');
data(:,:,2) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_3\NASplayerLOG_4-14-2013_20-7-27_M3.csv');
data(:,:,3) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_4\NASplayerLOG_4-14-2013_20-8-11_M4.csv');
data(:,:,4) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_5\NASplayerLOG_4-14-2013_20-8-41_M5.csv');
data(:,:,5) = tmpdata(1:3800,:);

%Iterate through all 5 data sets
for dIdx =1:5
```

```
    gpslat(:,dIdx) = data(:,2,dIdx);
    gpslon(:,dIdx) = data(:,3,dIdx);
    ekflat(:,dIdx) = data(:,4,dIdx);
    ekflon(:,dIdx) = data(:,5,dIdx);

    %RAW GPS (Old Format, must update if ever use
     xgps(:,dIdx) = abs(w-gpslon(:,dIdx))/lonpts;
     ygps(:,dIdx) = (n-gpslat(:,dIdx))/latpts;


    xekf(:,dIdx) = abs(w-ekflon(:,dIdx))/lonpts;
    yekf(:,dIdx) = (n-ekflat(:,dIdx))/latpts;
    %  figure(1);scatter(xekf(1),yekf(1),'g')
    %  figure(1);scatter(xekf(end),yekf(end),'r')
end
%      figure(1);scatter(xgps,ygps,'b')
%      figure(2);plot(xgps,ygps,'b')
pClr=['b' 'g' 'r' 'c' 'm'];
for pIdx =1:5
     figure(1);scatter(xekf(:,pIdx),yekf(:,pIdx),[pClr(pIdx) 'x'])
     figure(2);plot(xekf(:,pIdx),yekf(:,pIdx),pClr(pIdx),'LineWidth',1.5)
end



%Use Hemi data to make error boundaries
%Read the Hemisphere Data
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_1_SURVEY.csv');
hemiData(:,:,1) = hd(1:1200,:);    %V100 does not really collect at 10Hz, more like 5Hz.
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_2_SURVEY.csv');
hemiData(:,:,2) = hd(1:1200,:);
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_3_SURVEY.csv');
hemiData(:,:,3) = hd(1:1200,:);
[times hd] = readHemisphere('.\Data 13Apr\V100_POS_4_SURVEY.csv');
hemiData(:,:,4) = hd(1:1200,:);

for hIdx = 1:4
    hemiLat = hemiData(:,1,hIdx);
    hemiLat(isnan(hemiLat))=n; %shouldn't bias much
    hemiLon = hemiData(:,2,hIdx);
    hemiLon(isnan(hemiLon))=w; %shouldn't bias much
    %Use average to find center of error circle
    lonErrCtr(hIdx) = abs(w-mean(hemiLon))/lonpts;
    latErrCtr(hIdx) = (n-mean(hemiLat))/latpts;

    %For calculating error
%     avgHemiLon(hIdx) = mean(hemiLon);
%     avgHemiLat(hIdx) = mean(hemiLat);

end

%plot boxes to show error threshold of r

% r = 2; %2 meter error threshold
% %Convert to degrees->pixels
% rlon = (r/LONMETERS)/lonpts; %r lon pixels
% rlat = (-r/LATMETERS)/latpts; %r lat pixels
%
% outerBox = [lonErrCtr(1)+rlon,latErrCtr(1)-rlat; lonErrCtr(2)+rlon,latErrCtr(2)+rlat; lonErrCtr(3)-rlon,latErrCtr(3)+rlat; lonErrCtr(4)-
rlon,latErrCtr(4)-rlat; lonErrCtr(1)+rlon,latErrCtr(1)-rlat];
% innerBox = [lonErrCtr(1)-rlon,latErrCtr(1)+rlat; lonErrCtr(2)-rlon,latErrCtr(2)-rlat; lonErrCtr(3)+rlon,latErrCtr(3)-rlat;
lonErrCtr(4)+rlon,latErrCtr(4)+rlat; lonErrCtr(1)-rlon,latErrCtr(1)+rlat];
% figure(1); plot(outerBox(:,1),outerBox(:,2),'k','LineWidth',2);
% figure(1); plot(innerBox(:,1),innerBox(:,2),'k','LineWidth',2);


%Read the Hemisphere Data
[times hd] = readHemisphere('.\Data 13Apr\V100_MOTION_2.csv');
hemiLat = hd(:,1);
hemiLon = hd(:,2);
```

```
xHemi = abs(w-hemiLon)/lonpts;
yHemi = (n-hemiLat)/latpts;
figure(1);scatter(xHemi,yHemi,'y');grid on
title('5 Motion Tests vs.  Hemisphere V100')
legend('TEST 1', 'TEST 2','TEST 3', 'TEST 4','TEST 5','V100');
xlabel('Longitude');
ylabel('Latitude');

figure(2);plot(xHemi,yHemi,'y','LineWidth',1.5);grid on
title('5 Motion Tests vs.  Hemisphere V100')
legend('TEST 1', 'TEST 2' ,'TEST 3','TEST 4','TEST 5','V100');
xlabel('Longitude');
ylabel('Latitude');
% %
% %
% % %Error in meters
% % for eIdx = 1:4
% %     ekflonerr = mean(abs(ekflon(:,eIdx)-avgHemiLon(eIdx))) * LONMETERS;
% %     ekflaterr = mean(abs(ekflat(:,eIdx)-avgHemiLat(eIdx))) * LATMETERS;
% %     errekf(eIdx) = sqrt(ekflaterr^2 + ekflonerr^2)
% %     gpslonerr = mean(abs(gpslon(:,eIdx)-avgHemiLon(eIdx))) * LONMETERS;
% %     gpslaterr = mean(abs(gpslat(:,eIdx)-avgHemiLat(eIdx))) * LATMETERS;
% %     gpserr(eIdx) = sqrt(gpslaterr^2 + gpslonerr^2)
% %
% % end
```

B.4 Plot5PassHdg Analysis Script

This MATLAB script plots the headings collected during 5 passes of NAS data against the heading collected by the V100 for comparison purposes

B.4.1 Plot5PassHdg.m

```
close all;  %will close figures open until now
clear;  %clear vars


%Load the data from the rover NAS

%Read 5 motion test from NAS
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_1\NASplayerLOG_4-23-2013_3-8-17_M1_fixed360.csv');
data(:,:,1) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_2\NASplayerLOG_4-23-2013_3-7-8_M2_fixed360.csv');
data(:,:,2) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_3\NASplayerLOG_4-23-2013_3-9-48_M3_fixed360.csv');
data(:,:,3) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_4\NASplayerLOG_4-23-2013_3-10-56_M4_fixed360.csv');
data(:,:,4) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_5\NASplayerLOG_4-23-2013_3-11-53_M5_fixed360.csv');
data(:,:,5) = tmpdata(1:3800,:);


data(data>320)=data(data>320)-360;


%Iterate through all 5 data sets,must downsample to get hemi to match up
for dIdx =1:5
    ekfhdg(:,dIdx) = data(1:3:end,14,dIdx);
end
%      figure(1);scatter(xgps,ygps,'b')
%      figure(2);plot(xgps,ygps,'b')
pClr=['b' 'g' 'r' 'c' 'm'];
figure;
sampTimes = 0.05.*(1:3:3800);  %It looks like it is running at 20 Hz , why?!?
%
for pIdx =1:5
```

```
    plot(sampTimes,ekfhdg(:,pIdx),pClr(pIdx),'LineWidth',1.5);hold on;
end

%Read the Hemisphere Data
[times hd] = readHemisphere('.\Data 13Apr\V100_MOTION_3.csv');


hd(hd>320)=hd(hd>320)-360;


%must interpolate to make it match the hemidata, linear

plot(sampTimes,hd(1:length(ekfhdg),3),'k','LineWidth',1.5);

grid on;
title('Heading Data from 5 Tests vs.  Hemisphere V100')
legend('TEST 1', 'TEST 2','TEST 3', 'TEST 4','TEST 5','V100');
xlabel('Sample Time (s)');
ylabel('Heading (degrees)');
```

B.5 PlotVel Analysis Script

This MATLAB script plots the velocity magnitude data from one pass of NAS data against the V100 for comparison purposes

B.5.1 PlotVel.m

```
close all;  %will close figures open until now
clear;  %clear vars


%Load the data from the rover NAS

%Read 5 motion test from NAS
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_1\NASplayerLOG_4-14-2013_20-6-20_M1.csv');
data(:,:,1) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_2\NASplayerLOG_4-14-2013_20-6-47_M2.csv');
data(:,:,2) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_3\NASplayerLOG_4-14-2013_20-7-27_M3.csv');
data(:,:,3) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_4\NASplayerLOG_4-14-2013_20-8-11_M4.csv');
data(:,:,4) = tmpdata(1:3800,:);
tmpdata = xlsread('.\Data 13APR\NAS_MOTION_5\NASplayerLOG_4-14-2013_20-8-41_M5.csv');
data(:,:,5) = tmpdata(1:3800,:);

%Iterate through all 5 data sets
for dIdx =1:5
   gpsvelx(:,dIdx) = data(:,6,dIdx);
   ekfvelx(:,dIdx) = data(:,7,dIdx);
   gpsvely(:,dIdx) = data(:,8,dIdx);
   ekfvely(:,dIdx) = data(:,9,dIdx);
end

%X,YNot very interesting,only use for this is possible slip detection, get
%magnitude
gpsVelMag = sqrt(gpsvelx.^2+gpsvely.^2);
ekfVelMag = sqrt(ekfvelx.^2+ekfvely.^2);


%      figure(1);scatter(xgps,ygps,'b')
%      figure(2);plot(xgps,ygps,'b')
pClr=['b' 'g' 'r' 'c' 'm'];
figure;
sampTimes = 0.05.*(1:3800);  %It looks like it is running at 20 Hz , why?!?
%
```

```
for pIdx =1
    plot(sampTimes,gpsVelMag(:,pIdx),pClr(pIdx),'LineWidth',1.5);hold on;
    plot(sampTimes,ekfVelMag(:,pIdx),pClr(pIdx+1),'LineWidth',1.5);
end

grid on;
title('Velocity Estimation GPS vs.  EKF')
legend('GPS', 'EKF');
xlabel('Sample Time (s)');
            ylabel('Velocity (m/s)');
```

B.6 PlotGPSData Analysis Script

This MATLAB script plots NAS data collected during a single test pass against V100 data. It plots both EKF derived position and GPS only position. This script was used to evaluate data taken with a GPS signal loss

B.6.1 PlotGPSData.m

```
close all;  %will close figures open until now
clear;  %clear vars


%CONSTANTS
%distance from http://www.csgnetwork.com/degreelenllavcalc.html
LATMETERS = 111027.86; %METERS PER DGREE LAT
LONMETERS= 85829.94;%METERS PER DGREE LON
LATCORRECTION = 11;%meters, + move MAP north - move MAP south
LONCORRECTION = 0;%meters, + move MAP east - move MAP west

%Calcutlate dgree offset to adjust map
latoffset = LATCORRECTION/LATMETERS;
lonoffset = LONCORRECTION/LONMETERS;




% immap = imread('./maps/backyard.jpg');
immap = imread('./maps/ColPrkLotClose.jpg');



for i=1:2;  %make 2 figures
   figure; image(immap);hold on;
   % coords = load('./maps/backyard.txt');  %Read coordinate vector in
   % [n,s,e,w] format
   coords = load('./maps/ColPrkLotClose.txt');  %Read coordinate vector in [n,s,e,w] format
   n=coords(1)+latoffset;
   s=coords(2)+latoffset;
   e=coords(3)+lonoffset;
   w=coords(4)+lonoffset;
   latpts = (n-s)/size(immap,1);
   lonpts = (e-w)/size(immap,2);
   yticks = n - latpts .* get(gca,'YTick');
   latTickLabs = cell(size(yticks));
   for yidx = 1:length(yticks)
      latTickLabs{yidx} = num2str(yticks(yidx),'% 2.4f');
   end
```

```
    xticks = w + lonpts * get(gca,'XTick');
    lonTickLabs = cell(size(xticks));
    for xidx = 1:length(xticks)
        lonTickLabs{xidx} = num2str(xticks(xidx),'% 2.4f');
    end

    set(gca,'YTickLabel',latTickLabs);
    set(gca,'XTickLabel',lonTickLabs);

end
%REPLACE THIS WITH CUSTOM READ SO WE CAN TIME SYNC ETC...
data = xlsread('.\Data 13APR\NASplayerLOG_4-15-2013_3-35-45_sigloss_20hz.csv');

%full format
gpslat = data(:,2);
gpslon = data(:,3);
ekflat = data(:,4);
ekflon = data(:,5);
%Short rover format (check, may be wrong)
% ekflat = data(:,5);
%  ekflon = data(:,6);

%RAW GPS (Old Format, must update if ever use
 xgps = abs(w-gpslon)/lonpts;
 ygps = (n-gpslat)/latpts;
 figure(1);scatter(xgps,ygps,'b')
 figure(2);plot(xgps,ygps,'b','LineWidth',1.5)


xekf = abs(w-ekflon)/lonpts;
yekf = (n-ekflat)/latpts;
figure(1);scatter(xekf,yekf,'m*')
figure(2);plot(xekf,yekf,'m','LineWidth',1.5)
%  figure(1);scatter(xekf(1),yekf(1),'g')
%  figure(1);scatter(xekf(end),yekf(end),'r')

%Read the Hemisphere Data
[times hd] = readHemisphere('.\Data 13Apr\V100_MOTION_3.csv');
hemiLat = hd(:,1);
hemiLon = hd(:,2);

xHemi = abs(w-hemiLon)/lonpts;
yHemi = (n-hemiLat)/latpts;

figure(1);scatter(xHemi,yHemi,'c');grid on;
title('Lost Signal (at Corner 3) vs.  Hemisphere V100')
legend('GPS only','NAS','V100');
xlabel('Longitude');
ylabel('Latitude');

figure(2);plot(xHemi,yHemi,'c','LineWidth',1.5);grid on;
title('Lost Signal (at Corner 3) vs.  Hemisphere V100')
legend('GPS only','NAS','V100');
xlabel('Longitude');
ylabel('Latitude');
```

B.6.2 PlotStaticHeading.m

This MATLAB script plots heading data taken while the NAS was sitting static versus data taken from the same location and orientation by the V100 for comparative purposes.

B.6.3 PlotStaticHeading.m

```
close all;  %will close figures open until now
clear;  %clear vars


%Load the data from the rover NAS

%Read 4 static test points from NAS
tmpdata = xlsread('.\Data 13APR\NAS_STATIC_1\NASplayerLOG_4-23-2013_3-20-6_S1_fixed.csv');
data(:,:,1) = tmpdata(1:6000,:); %take approx 5 min
ekfhdg = data(1:3:end,14,1);
times=0.05*(1:3:6000);
figure; plot(times,ekfhdg,'b','LineWidth',1.5);
hold on;grid on;





%Use Hemi data to make error circles
%Read the Hemisphere Data
[ts hd] = readHemisphere('.\Data 13Apr\V100_POS_1_SURVEY.csv');
hemiData(:,:,1) = hd(1:length(ekfhdg),:);    %V100 does not really collect at 10Hz, more like 5Hz.
plot(times,hemiData(:,3,1),'g','LineWidth',1.5);
title('Static Heading Estimation Comparison of the NAS and V100');grid on

legend('NAS', 'V100');
xlabel('Time (s)');
ylabel('Heading (degrees)');
```

B.7 readHemisphere Analysis Support Function

This MATLAB function supports all the previously listed scripts by parsing data packets from the Hemisphere V100 GPS.

B.7.1 readHemisphere.m

```
function [timeArray hemiData] = readHemisphere( filename )
%READHEMISPHERE read the GPGGA and GPHDT sentences and kick out a matrix
%with the pertinent data
fHemiLog = fopen(filename,'r','n','US-ASCII');
hemiData = fgets(fHemiLog);
GGAIdx = 1;
HDTIdx = 1;
while ischar(hemiData)
    %Is GPGGA? - Location
    if strfind(hemiData,'GPGGA')
        [time,r]=strtok(hemiData,','); %timestamp - discard for now
%         [~,r]=strtok(hemiData,','); %timestamp - discard for now
        [~,r]=strtok(r,','); %GPGGA - discard
        [~,r]=strtok(r,','); %UTC time - discard
        [tmplat,r] = strtok(r,','); %lat -
        [~,r]=strtok(r,','); % N - discard
        [tmplon,r] = strtok(r,','); %lon -
        [~,r]=strtok(r,','); %W -
        [tmpquality,r] = strtok(r,','); %quality to check if good
        [tmpnumsats,r] = strtok(r,','); %# of sats used
        [tmpHDOP,r] = strtok(r,','); %HDOP, measure of qulaity
        [tmpaltitude,r] = strtok(r,','); %alt
        %discard rest
        %convert to numbers and store
        lat(GGAIdx) = str2double(tmplat);
        lon(GGAIdx) = -str2double(tmplon); %lon - (FOR W (all of US) it's negative)
        quality(GGAIdx) = str2double(tmpquality); %quality to check if good
        numsats(GGAIdx) = str2double(tmpnumsats); %# of sats used
        HDOP(GGAIdx) = str2double(tmpHDOP); %HDOP, measure of qulaity
        altitude(GGAIdx) = str2double(tmpaltitude); %alt
        time=strtok(time,'<DataTime>');
        timeArray(GGAIdx) = cellstr(time);
        GGAIdx = GGAIdx +1;
    %Is GPHDT? - Heading
    elseif strfind(hemiData,'HEHDT')
        [time,r]=strtok(hemiData,','); %timestamp - discard for now
        [~,r]=strtok(r,','); %HEHDT - discard
        [tmphdg,r]=strtok(r,','); %heading - no indication of qulaity...
        %discard rest
        hdg(HDTIdx) = str2double(tmphdg);
        HDTIdx = HDTIdx +1;
    else
        disp('Log Parse Error');
    end
    hemiData = fgets(fHemiLog);
end
%FORMAT the vectors into a matrix that matches the data from the NAS
if length(hdg) > length(lat)
    hdg = hdg(1:length(lat));
else
    lat = lat(1:length(hdg));
    lon = lon(1:length(hdg));
    timeArray = timeArray(1:length(hdg));
end
%Convert to decimal degrees........


lat = fix(lat/100)+rem(lat/100,1)*100/60;
lon = fix(lon/100)+rem(lon/100,1)*100/60;

hemiData = [lat',lon',hdg']; %Just the basics for now
timeArray = timeArray';

end
```

B.8 readNAS Analysis Support Function

This MATLAB function supports all of the analysis scripts by parsing the NAS data packets,

B.8.1 readNAS.m

```
function [timeArray hemiData] = readNAS( filename )
%READHEMISPHERE read the GPGGA and GPHDT sentences and kick out a matrix
%with the pertinent data
fHemiLog = fopen(filename,'r','n','US-ASCII');
hemiData = fgets(fHemiLog);
GGAIdx = 1;
HDTIdx = 1;
while ischar(hemiData)
    %Is GPGGA? - Location
    if strfind(hemiData,'GPGGA')
        [time,r]=strtok(hemiData,','); %timestamp - discard for now
%         [~,r]=strtok(hemiData,','); %timestamp - discard for now
        [~,r]=strtok(r,','); %GPGGA - discard
        [~,r]=strtok(r,','); %UTC time - discard
        [tmplat,r] = strtok(r,','); %lat -
        [~,r]=strtok(r,','); % N - discard
        [tmplon,r] = strtok(r,','); %lon -
        [~,r]=strtok(r,','); %W -
        [tmpquality,r] = strtok(r,','); %quality to check if good
        [tmpnumsats,r] = strtok(r,','); %# of sats used
        [tmpHDOP,r] = strtok(r,','); %HDOP, measure of qulaity
        [tmpaltitude,r] = strtok(r,','); %alt
        %discard rest
        %convert to numbers and store
        lat(GGAIdx) = str2double(tmplat);
        lon(GGAIdx) = -str2double(tmplon); %lon - (FOR W (all of US) it's negative)
        quality(GGAIdx) = str2double(tmpquality); %quality to check if good
        numsats(GGAIdx) = str2double(tmpnumsats); %# of sats used
        HDOP(GGAIdx) = str2double(tmpHDOP); %HDOP, measure of qulaity
        altitude(GGAIdx) = str2double(tmpaltitude); %alt
        time=strtok(time,'<DataTime>');
        timeArray(GGAIdx) = cellstr(time);
        GGAIdx = GGAIdx +1;
    %Is GPHDT? - Heading
    elseif strfind(hemiData,'HEHDT')
        [time,r]=strtok(hemiData,','); %timestamp - discard for now
        [~,r]=strtok(r,','); %HEHDT - discard
        [tmphdg,r]=strtok(r,','); %heading - no indication of qulaity...
        %discard rest
        hdg(HDTIdx) = str2double(tmphdg);
        HDTIdx = HDTIdx +1;
    else
        disp('Log Parse Error');
    end
    hemiData = fgets(fHemiLog);
end
%FORMAT the vectors into a matrix that matches the data from the NAS
if length(hdg) > length(lat)
    hdg = hdg(1:length(lat));
else
    lat = lat(1:length(hdg));
    lon = lon(1:length(hdg));
    timeArray = timeArray(1:length(hdg));
end
%Convert to decimal degrees........


lat = fix(lat/100)+rem(lat/100,1)*100/60;
lon = fix(lon/100)+rem(lon/100,1)*100/60;

hemiData = [lat',lon',hdg']; %Just the basics for now
timeArray = timeArray';

end
```