Development of ANVEL HIL/SIL Simulation Environment for Rapid Prototyping of Navigation Algorithms

by

Brently Nelson

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 5, 2018

Keywords: Hardware-In-the-Loop, Software-In-the-Loop, GPS/INS, Navigation Units

Approved by

David Bevly, Chair, Professor of Mechanical Engineering
Dan Marghitu, Professor of Mechanical Engineering
John Hung, Professor of Electrical and Computer Engineering

Abstract

This thesis presents the development of a Hardware In the Loop/Software In the Loop (HIL/SIL) simulation environment with the purpose of testing Global Positioning System/Inertial Navigation System (GPS/INS) navigation units in ANVEL. HIL/SIL test beds are widely popular research tools that allow researchers to combine high fidelity vehicle simulations with errors inherent to hardware implementation, providing more realistic simulations. Researchers may therefore use these test beds further along in the design process with confidence the simulation is behaving as in the real-world, removing operators from the test environment until final stages of development when the technology is more fully proven. The work presented in this thesis focuses on the development of a modular HIL/SIL test bed using software and hardware add-ons to the Autonomous Navigation Virtual Environment Laboratory (ANVEL), a high fidelity vehicle simulation environment. The test bed is designed with the goal of testing GPS/INS navigation units, specifically the unit developed by Auburn's GAVLab.

The capabilities of ANVEL are extended through the use of a plugin to relay vehicle state information out of the simulation environment. A second plugin is developed for software GPS simulation in which satellites are simulated using broadcast ephemeris data. Ray tracing is made possible through ANVEL's physics engine, allowing the simulation to detect satellite obstructions. Vehicle state information from ANVEL is used to generate Inertial Measurement Unit (IMU) and Wheel Speed Sensor (WSS) data in software modules which corrupt the information according to error models derived from real-world sensors. Hardware implementations for the GPS, IMU, and WSS modules are then developed to add realism to the test environment. A Spectracom GPS simulator is used to provide Radio Frequency (RF) signal in real-time to the navigation unit using position and satellite availability information from ANVEL. A serial interface is then developed such that the IMU module outputs a serial signal to emulate the real sensor. Finally, quadrature signals are generated using Pulse Width Modulators (PWMs) to represent encoder pulses from wheel speed sensors. Each of the developed software and hardware

modules are then validated in both static and dynamic test scenarios using error characteristics from sensors in the GAVLab navigation unit as benchmarks for comparison. Results from validation demonstrate the test bed is capable of outputting realistic sensor measurements which may be used interchangeably with sensors in a navigation unit for the purpose of algorithm development.

Acknowledgments

There are many people I would like to thank for helping me get to this stage in my education and my life. First and foremost, I would like to thank my parents Gene and Leah. From an early age they instilled in me that nothing but my best was acceptable and that mindset has helped me accomplish many things. I would also like to thank my sister Cadie for always being there for me and always lending an ear when I needed someone to talk to. Without the love and support of my family, I would never have made it this far in my life or education. I would also like to thank my extended family around Auburn for always being there with a warm meal and a kind word whenever I needed.

I would especially like to thank Dr. David Bevly for providing me with the opportunity to pursue my graduate degree in the GAVLab. My time in graduate school has allowed me to do so many things and live in places I never imagined I would get to and I cannot thank Dr. Bevly enough. I also want to thank Dr. John Hung and Dr. Dan Marghitu for their oversight of this thesis. Additionally, I would like to thank Dr. Scott Martin for introducing me to the lab and asking me to do undergraduate research. Without Dr. Martin, I would never have worked in the GAVLab and missed out on so many opportunities. Thanks also goes to all the members of the GAVLab who helped keep me sane during graduate school. They were always open to having a good time and especially a game or two of ping pong. Special thanks goes out to Dan Pierce, Scott Martin, Grant Apperson, and Chris Rose for always being willing to provide ideas and help related to my research. Finally, I would like to give thanks to the Lord for all the opportunities and people He has given me thus far that led me to this point.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There has been much work in recent years developing Hardware In the Loop/Software In the Loop (HIL/SIL) simulation environments due to their ability to aid in rapid development and testing of navigation and control algorithms. The increased usage of Unmanned Ground Vehicles (UGVs) and Unmanned Aerial Vehicles (UAVs) in both civilian and military applications are a large driving force behind the adaptation of HIL/SIL test beds. Because these systems allow for more realism, researchers can use simulation techniques much further along in the design process with confidence the simulation is behaving similar to the real-world. This ability removes operators from the testing environment in the early stages of development when the technology may not be fully proven; thus greatly increasing safety during testing. With the HIL/SIL environment, a human test operator is absent until the final stages of algorithm development. HIL/SIL testing also allows researchers to investigate edge case scenarios without risking operator safety or costly test vehicles. Without as much need for costly field testing, researchers may test algorithms more thoroughly. Also, researchers may easily analyze the effects of changes in algorithms, sensor error characteristics, and environmental changes through repeatable test runs in a controlled environment. In many applications, a navigation unit containing common navigation sensors such as Global Positioning System (GPS), Inertial Measurement Units (IMUs), Wheel Speed Sensors (WSS), and a navigation algorithm is outfitted to a vehicle with the purpose of localization for vehicle autonomy functions. The navigation

algorithm in these navigation units involves GPS and an Inertial Navigation System (INS) integrated into a GPS/INS navigation solution. As autonomous vehicle technologies progress, there is an increased need for more accurate and realistic testing of these GPS/INS navigation units in a laboratory environment.

1.2 Prior Research

GPS/INS algorithms are popular tools employed in autonomous vehicle localization due to the complementary nature of the two technologies. Robust and accurate navigation solutions can be achieved when using high quality sensors. An in depth overview of the sensors, sensor characteristics, and various GPS/INS algorithm options is found in [25]. Salmon explored variations of GPS/INS algorithms and performance when aiding low-cost vehicle sensors with a standalone vehicle model in [48]. In [38], the author presented an in depth look at the advantages of GPS/INS aided with Dynamic Real Time Kinematic (DRTK) for use in automated convoys.

A wealth of information on the sensor systems involved in GPS/INS localization may be found in [39] and [25]. Powell detailed the development of a MATLAB GPS simulator using satellite ephemeris and atmospheric data in [43]. Wall and Flenniken developed simple sensor models to approximate inertial sensor behavior in [58] and [16]. Both Wall and Flenniken demonstrate the viability of the simple sensor models via the experimental identification techniques of Allan variance [1] and autocorrelation.

There has been much work in recent years developing HIL/SIL test beds as vehicle autonomy has become more prevalent. At the base of these test beds are high fidelity vehicle simulation environments capable of simulating complicated vehicle physics and terrain interactions. An exploration into the various simulation environments typically employed in UAV/UGV HIL/SIL systems can be found in [15]. The author examines strengths and weaknesses of many popular robotics simulation environments such as ANVEL and Gazebo, stating that no one environment fully covers all the needs of researchers.

Gazebo is popular among unmanned systems developers because it is built into the Robot Operating System (ROS) environment, a tool widely used in the field of robotics. Gazebo

employs the same physics and graphics engines as ANVEL. Odelga uses Gazebo for a multi-UAV simulation using computational units onboard UAVs to test algorithmic computational feasibility as well as inter-UAV communications in [42]. Swanson also made use of ROS and Gazebo for development of a HIL driving simulator and performs an analysis comparing the HIL simulator against traditional simulator implementations in [56]. Other authors have used the ROS environment outside of the Gazebo simulation. Yan employed the Modular OpenRobotics Simulation Engine (MORSE) within a ROS based testbed for the purpose of realistic multi-robot simulations in [61]. Several unmanned ground vehicle simulators were evaluated in [31], in which the authors chose ANVEL for future development and extended it for use with ROS. The US Army currently uses ANVEL with ROS in its HIL/SIL environments for unmanned ground vehicle projects such as the Autonomous Ground Resupply (AGR) [41] and Wingman [50, 49].

The work presented in this thesis also employs ANVEL with ROS and is intended to integrate into the current AGR HIL/SIL system at US Army Tank Automotive Research, Development and Engineering Center (TARDEC). The current system uses ANVEL for vehicle dynamics and sensor simulation and an additional Linux PC that runs the By-Wire Active Safety Kit (BWASK) and Autonomy Kit (AK) software. The HIL/SIL also employs ROS and is mainly used to simulate autonomous vehicle convoys. The AK uses the vehicle and sensor simulation from ANVEL to make all autonomy decisions for the follower vehicles in the convoy including teleoperation and path following, while the BWASK physically controls the vehicle hardware. TARDEC's HIL/SIL uses a simulated radar on each vehicle which passes obstacle information to the AK software. The HIL/SIL also has a steering column, pedals, and transmission selector box integrated into the system to allow researchers to drive the lead vehicle in the convoy if desired, or the path may be simulated. The current system is capable of simulating a six vehicle convoy but does not have the ability to employ GPS/INS navigation units to obtain a realistic global navigation solution, which this works adds.

Steffes presented the development of a reconfigurable HIL test bench for the SHEFEX2 mission in [53]. The developed HIL employed SIL through high fidelity MATLAB/Simulink

models implemented on a dSPACE real-time simulator and used a rotation table to stimulate inertial sensors. Like the work presented in this thesis, Steffes employs a real-time GPS simulator to provide GPS signal to the HIL/SIL. In [29], SIL testing was used for a UAV with GPS/INS localization and a (Light Detection and Ranging) LiDAR sensor for Simultaneous Localization and Mapping (SLAM) applications. Lepej used HIL with UAVs for the purpose of SLAM in GPS denied environments [37]. This work does not employ ranging sensors such as LiDAR and therefore is not suited for SLAM applications, but may be extended for such situations. The FALTER program employed model-based software in the loop testing for UAVs designed for indoor use, such as exploration of a factory after an accident [3].

There are also many examples of HIL/SIL test environments employed in the auto industry. As vehicles become more complex, HIL/SIL allows for ease of integration between multiple Advanced Driver-Assistance Systems ADAS systems. Galko presented a vehicle HIL system for prototyping and validation of ADAS in order to evaluate new algorithms and sensors on a complete vehicle [20]. Like Galko, a full scale ADAS equipped vehicle was implemented on a chasis dynomometer in [23]. In [47], a heavy vehicle HIL crash avoidance safety system is introduced and validated experimentally. King discussed the usage of HIL test systems to support various stages of development of vehicle system electrical architecture and Electrical Control Units (ECU) in [35]. Draper Laboratory employed a HIL test bed to analyze the performance of advanced GPS/INS algorithms in degraded GPS environments including jamming scenarios in [17]. This thesis is intended to be more flexible with hardware and software elements that can be removed, as well as allow testing with GPS/INS units from multiple vendors.

1.3 Contributions

This thesis details the development of a HIL/SIL simulation environment specializing in the development, testing, and comparison of GPS/INS navigation systems. Outputs from the simulation environment are then compared to outputs from the sensors being simulated to verify system performance. The contributions this thesis makes to the field of study are as follows:

- Integration of the ANVEL simulation environment with GPS/INS navigation units.

- Realistic ray tracing capabilities in GPS simulations used in conjunction with ANVEL.
- Software and hardware modules capable of outputting realistic sensor measurements compatible with various navigation units.
- Test bed with a modular system architecture to be used as the base of further HIL/SIL development through addition of different sensor models and technologies.

1.4 Thesis Outline

In Chapter 2, the basics of GPS/INS navigation are introduced with an emphasis on the sensors involved and their respective error sources. Chapter 3 details the design of the simulation environment including measurement and error models used for each sensor. Both software and hardware sensor modules are introduced and implementation methods are discussed. In Chapter 4, the hardware sensors are characterized and used as a baseline for comparison with results obtained from the developed simulation environment. Chapter 5 presents conclusions and future work for the current implementation and future development. Additionally, the GPS satellite position calculation is included in the Appendix.

Chapter 2

GPS/INS Technology Introduction

Autonomous vehicles are typically outfitted with many different sensor systems including, but not limited to, GPS, IMU, wheel speed sensors, radar, lidar, and cameras. The work developed in this thesis focuses on autonomous vehicle navigation using GPS, IMUs, and wheel encoders but provides a framework for easy addition of any of the aforementioned sensor technologies. The purpose of this work is to allow for easy addition of these simulated sensor measurements into fused navigation algorithms such as GPS/INS. This chapter provides an introduction to the sensor systems typically used, system capabilities, and error sources as well as an introduction to navigation algorithms.

2.1 GPS

The Global Positioning System consists of the space, control, and user segments which combine to provide users with a position, velocity, and time solution. The space segment consists of at least 24, and generally 32, operational satellites in six orbital planes. The satellites, or Space Vehicles (SVs), are oriented such that a user with open sky has a clear line of sight to at least four SVs at any given time. The control segment, located in Colorado Springs, Colorado, operates monitoring stations all around the world and is tasked with maintaining satellite orbits and accurate GPS time. The control segment periodically sends time and orbit corrections to each satellite to maintain the accuracy of the transmitted signals. The user segment is composed of individual GPS receivers tracking the Radio Frequency (RF) signal broadcast by each satellite [19]. Each SV transmits RF signals on at least two frequencies, L1 and L2. The L1

frequency is 1575.42 MHz, while L2 is 1227.60 MHz. Each RF signal contains a ranging code, called the Coarse Acquisition (C/A) and P code, for civilian and military applications respectively, as well as a navigation message modulated on a sinusoidal carrier. The ranging codes are pseudo-random binary sequences unique to each satellite and called Pseudo-Random Noise (PRN) codes. Each PRN code is orthogonal to all others which allows the receiver to differentiate signals from all satellites broadcast on the same frequency through Code Division Multiple Access (CDMA) [25]. The C/A code is 1,023 chips long and repeats every millisecond, while the P code is 6.1871 x 10^{12} chips long and is broadcast at ten times the rate of the C/A code. The P code is encrypted, limiting access to only those users with the encryption key. However, beginning in 2005 with the launch of Block IIR-M satellites, the L2C ranging code is available, making dual frequency ranging measurements available to civilian users [13]. Civilian users may also access the L2 P coded signals through semi-codeless tracking techniques [51].

The navigation message broadcast by each satellite provides the user with satellite information and the signal transmission time [24]. The broadcast satellite information contains almanac data, satellite ephemeris data, and satellite health information which allow the user to determine the satellite's position and velocity. The satellite ephemeris is a set of quasi-Keplerian orbital parameters calculated by the GPS control station, with each SV broadcasting its own ephemeris parameters. The ephemeris consists of the 6 Keplerian orbital elements and nine terms designed to account for the changes in orbit over time due to pertubations [39]. The use of ephemeris data to calculate SV position and velocity is described in Appendix A. The distance from a given satellite to the receiver is then calculated using Time Of Flight (TOF) divided by the speed of light. With range information from three satellites, the user's position can be estimated, assuming a perfectly synchronized receiver clock. Adding the range from a fourth satellite allows for the receiver to solve for user position and the receiver clock error. The GPS receiver measurements of concern for this work are introduced in the following subsection with GPS errors and an introduction to the GPS Position, Velocity, and Time (PVT) solution in the following subsections respectively. The corresponding equations used to generate the measurements and PVT solution for this work are found in Section 3.3.

2.1.1 GPS Receiver Measurements

The GPS receiver generates a replica of each satellite's C/A code to use during satellite acquisition and range determination. A detailed explanation of the acquisition process can be found in [39]. In short, the receiver takes advantage of the fact that each satellite's C/A code is unique and orthogonal to all others by leveraging the correlation function. The incoming signal correlates with only one satellite's receiver generated C/A code, allowing the receiver to determine which satellite the signal is originating from. The incoming signal will be out of phase with the receiver generated replica, so the receiver must shift the replica until the correlation peak is found. The phase shift of the replica signal allows the deduction of the signal transmission time, due to the fact the C/A code is transmitted at known time intervals. The time of arrival of the signal is determined from the receiver clock, and differencing the time of transmission and arrival yields the signal transit time. Multiplying the signal transit time by the speed of light yields the most basic GPS measurement, the pseudorange [39, 25], shown in Equation (2.1)

$$\rho_k = (t_a - t_t)c \tag{2.1}$$

where t_a and t_t represent signal arrival and transmission times, respectively, and subscript k denotes the satellite. The pseudorange is the most basic measurement made by a GPS receiver and serves as the basis for the range for position determination. The pseudorange represented in Equation (2.1) is idealized and contains no errors. The errors present in a real pseudorange measurement are discussed in Section 2.1.2.

Correlation of the incoming signal with only the C/A code results in a sinusoidal signal, due to the carrier component of the signal. To resolve the data in the GPS signal, the incoming signal must also be multiplied by a replica of the carrier wave. The measured carrier is delayed from the reference carrier generated by the receiver. This delay is known as the carrier phase and is an indirect measurement measurement of the signal transit time. The carrier phase measurement is in terms of cycles of the carrier signal. The receiver measures the partial cycle, but has no way of knowing how many full cycles of the carrier signal have occurred between the satellite and receiver which is referred to as integer ambiguity. The receiver measures the

initial carrier phase and then tracks the phase change due to movement of the satellite and receiver. If the phase grows by a full wavelength by the next measurement epoch, the carrier phase measurement is now a full cycle plus the original fractional cycle. With no errors present, the carrier phase measurement is represented by Equation (2.2)

$$\phi(t) = \phi_u(t) - \phi^s(t_a - \tau) + N \tag{2.2}$$

where $\phi_u(t)$ is the phase of the receiver generated signal, $\phi^s(t_a - \tau)$ is the phase of the signal received from the satellite at time t_a, or the phase of the signal at the satellite at transmission time, while τ represents signal transit time, and N is the integer ambiguity. To represent the carrier phase in terms of user range, Equation (2.2) is simplified to Equation (2.3)

$$\phi(t) = \frac{r(t, t-t)}{\gamma} + N \tag{2.3}$$

where γ is the carrier wavelength and $r(t_a, t_a - \tau)$ is the geometric range between the user position at time t_a and the satellite position at time $t_a - \tau$. The change in the carrier phase measurement over time corresponds to changes in the user-satellite range and is referred to as the integrated Doppler or delta pseudorange. The rate of change of the carrier phase measurement gives the pseudorange rate [39].

Relative motion between the satellites and the receiver causes a shift in the carrier frequency due to the Doppler effect [39]. In order to maintain a carrier replica to multiply the incoming signal by, the receiver tracks the Doppler shift of each incoming signal. The Doppler shift is the difference between the nominal carrier frequency and the frequency of the received carrier signal. Doppler shift may be transformed into a pseudorange rate measurement using Equation (2.4)

$$\dot{\rho_k} = -\frac{c}{f_c}\Delta f_{c_k} \tag{2.4}$$

where f_c is the carrier frequency and Δf_{c_k} represents the difference between the received and nominal carrier frequency. The Doppler shift and pseudorange rate measurements are used by the receiver to calculate user velocity discussed in more detail in section 3.3.6

2.1.2 GPS Errors

A large source of error in the GPS range measurement is the satellite clock bias. The clock bias is such a large error source due to the fact the bias is multiplied by the speed of light when converting transit time to range. A one micro second bias corresponds to around 300 m of error [39]. The satellite clocks are extremely stable, but do drift very slowly over time. The clocks generally behave better without frequent corrections, so the clock drift is monitored by the control segment and a clock correction term is broadcast in the ephemeris data for each SV and used by the receiver to correct measured ranges. The other clock error present in GPS measurements is the receiver clock bias, however, this bias is estimated during the position solution calculation. The drift rate of the receiver and satellite clocks manifest as a bias in the measured Doppler shift of the incoming carrier signal. The error on the Doppler shift/pseudorange rate measurement due to satellite clock drift can be corrected using the satellite clock bias terms in the ephemeris. The receiver clock drift is treated as unknown during the velocity estimation rather than an error source. Because the Dopper shift is biased by the receiver clock bias rate, the receiver actually measures pseudorange rate. Along with satellite clock correction errors in the broadcast data are satellite ephemeris errors along the radial, along track, and cross track directions of the satellite orbit. Since the control segment estimates the satellite positions based on range measurements, the along track and cross track errors can be several times larger than the radial component. Fortunately, the error in pseudorange measurement is the projection of the satellite position error vector on the satellite receiver line of sight which depends mostly on the radial component of the ephemeris error. The control segment continuously tracks the combination of ephemeris and satellite clock errors within 1m rms and uploads data periodically to the satellites keeping the current estimates of range error due to ephemeris and clock errors to about 1.5m rms each [39].

Another major error source for GPS is atmospheric effects. GPS signals are refracted while travelling through the atmosphere, meaning the signal's velocity is changed [39]. This velocity change affects the signal transit time to the receiver for each satellite differently and therefore cannot be estimated as a bias like the receiver clock error. The first layer of the

atmopshere the signal passes through is the ionosphere. The ionosphere is a layer of ionized gases about 50-100 kilometers above the surface of the earth. The ionosphere is a dispersive medium, meaning the refractive index depends on the signal frequency. Due to this fact, as the signal passes through the ionosphere, a phenomena known as code-carrier divergence occurs, meaning the code and carrier exhibit different refractive indexes as a result of the difference in frequency. The C/A code is delayed and the carrier signal is advanced, meaning the receiver measures the code-based range too long and the carrier too short [39]. Ionospheric correction terms are broadcast in the satellite ephemeris and may be used by the receiver in an effort to correct the pseudorange. Receivers with the ability to make dual frequency measurements can take advantage of the fact the ionosphere is dispersive. By measuring the difference between each frequency's pseudorange, the receiver can estimate the ionospheric delay using Equation (2.5)

$$\begin{aligned} \delta\rho(f_1) &= 1.54573\rho_{12} \\ \delta\rho(f_2) &= 2.54573\rho_{12} \end{aligned} \tag{2.5}$$

where $\delta\rho$ is the pseudorange error for each frequency represented by f_1 and f_2 respectively and ρ_{12} represents the difference between the L1 and L2 pseudoranges [27].

After the ionosphere, the signal then passes through the troposphere. The troposphere is the lower part of Earth's atmosphere and consists of dry gases and water vapor. Unlike the ionosphere, the troposphere is not a dispersive medium and therefore delays the code and carrier by the same amount. The troposphere delay comprises of the wet and dry delay. The wet delay is due to the water vapor present in the troposhphere, is highly variable, and hard to predict. However, the dry delay is due to the dry gases present, is easily modeled, and represents roughly 90% of the tropospheric delay.

Range error due to the atmospheric effects increases as elevation angle decreases due to the fact the GPS signal must travel through more of the atmosphere when the satellite is at lower elevations and is therefore delayed more as seen in Figure 2.1. When modeling atmospheric errors, a mapping function is used to represent the fact the signal is delayed more at lower

elevation angles scaling the zenith delay as a function of elevation angle. The Root Mean Square (RMS) residual range error due to atmospheric models at mid latitudes is about 5m, but can vary greatly greatly due to receiver location, satellite elevation angles, and the state of the ionosphere [39]. The models used to calculate atmospheric delay, mapping functions for each section of the atmosphere, and pseudorange correction using the atmospheric models are discussed further in Section 3.3.2.

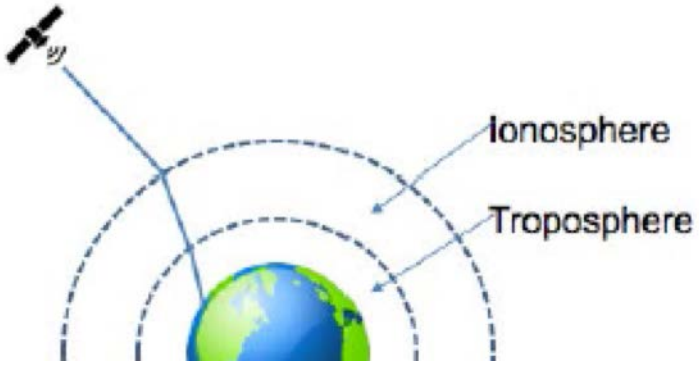

Figure 2.1: Visualization of Atmospheric Layers.

The code and carrier measurements made by the receiver are also subject to random measurement noise. Receiver measurement noise covers RF radiation sensed by the antenna unrelated to the GPS signal, noise introduced by the antenna, amplifiers, cables, receiver, interference from other GPS signals, and quantization noise. Without the presence of interference, the receiver will effectively see a waveform comprised of the GPS signal and random noise. The receiver noise varies with satellite signal strength which varies with satellite elevation angle. The rms pseudorange error for receiver noise is typically 0.25-0.5m [39].

Multipath error is due to the signal from one satellite reaching the receiver via more than one path. Typically, the antenna receives the line of sight signal from the satellite and one or more reflections of the signal from nearby structures or the ground. The reflected signal travels further than the line of sight signal and is therefore delayed in time corresponding to this extra distance. The resulting code and carrier phase measurements made by the receiver represent the sum of the received signals. The carrier phase multipath error is typically two orders of magnitude smaller than the code phase error and is no worse than a quarter carrier cycle. Error in pseudorange measurements due to multipath typically varies from 1m in a

benign environment to 5m in a highly reflective environment [39]. Table 2.1 summarizes typical pseudorange measurement errors for a single frequency receiver.

Table 2.1: Typical Pseudorange Measurement Errors for Single Frequency Receiver [39]

Error Source	RMS Range Error
Satellite Clock and Ephemeris	3 m
Atmospheric Errors	5 m
Receiver Noise and Multipath	1 m

2.1.3 GPS Position Solution

The GPS position solution caluclation is based on a process called trilateration. Using trilateration, if the position of an object and distance from that object are known at the same instance in time, the user's position can be calculated in the same coordinate frame, provided there are at least as many measurements as degrees of freedom [4]. A straightforward 2-D example of trilateration is presented in [39]. If a user measures the distance to a station with known location, that user must be located at some point on a circle with the radius equal to the measured distance from the station. The addition of a second range measurement from a second station with known position reduces the user's possible position to two locations where the circles intersect. The user may be able to reject one of these positions if sufficient prior information is known, otherwise a third measurement is needed. The mathematical model of the 2-D range to each station is given in Equation (2.6)

$$r_k = \sqrt{(x_k - x)^2 + (y_k - y)^2} \tag{2.6}$$

where (x_k, y_k) are the known station coordinates and r_k is the range to each respective station. Solving the set of quadratic equations represented by Equation (2.6) with measurements from at least two stations yields the user's 2-D position (x, y)[39].

The extension to 3-D positioning is relatively straightforward provided one of the range measurements comes from a station with a high elevation angle. The requirement of a station with high elevation angle is not an issue with GPS due to the fact the sations are satellites located in space and the design of the orbits. Three measurements from satellites provide the

user with two possible locations, one on or near the surface of the earth, and the other roughly 40,000 km in space. Obviously, the user can discard the solution in space. Due to the fact that the GPS receiver clock is not synchronized with the satellite clocks, one additional measurement is required. As previously stated, the satellite information broadcast by each satellite specifies the time of transmission of the signal. Assuming the satellite clock and receiver clock are synchronized, the receiver may calculate the range to each respective satellite by dividing the difference between signal reception and transmission times by the speed of light. However, due to cost constraints, the receiver clock must be inexpensive and therefore will not be synchronized with GPS time and will be biased relative to the satellite clocks. This bias leads to the range to each satellite being either too long or short. To deal with the receiver clock bias, one additional satellite must be used, bringing the number of required observations to four to estimate the 3-D position of the user and the user's clock bias [39]. The mathematical model represented by Equation (2.6) is extended to three dimensions in Equation (2.7).

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} = r_k \tag{2.7}$$

The errors due to clocks, atmosphere, and receiver noise are then added to the range, yielding the pseudorange. The 3-D representation of pseudorange including errors is found in Equation 2.8

$$\sqrt{(x_k - x)^2 + (y_k - y)^2 + (z_k - z)^2} + c[\delta t_u - \delta t^s] + I_\rho + T_\rho + \epsilon\rho = \rho_k \tag{2.8}$$

where I represents ionosphere error, T represents troposphere error, and ϵ is noise and unmodeled errors present in the signal. The satellite clock bias is δt^s and the receiver clock bias is δt_u. Application of the pseudorange in solving for user position can be seen in Section 3.3.6.

2.2 Inertial Navigation Systems

Inertial Navigation Systems (INS) are comprised of inertial sensors and a navigation processor. The inertial sensors typically employed are accelerometers and gyroscopes. Accelerometers measure the specific force applied along its sensitive axis, which can be converted to an

acceleration, while gyroscopes measure the angular velocity of the body on which it is mounted along its sensitive axis. Because neither of these sensors provides a global measurement, an INS can only provide a navigation solution relative to an initial position. The typical INS is comprised of three accelerometers and three gyroscopes, with each accelerometer and gyroscope mounted orthogonaly to the other sensors of the same type. Mounting the sensors in such a way provides means for measurements along all six degrees of freedom of the body the INS is mounted on. The navigation processor uses measurements from the accelerometers and gyroscopes to compute a navigation solution reference from the initial position. Accelerometers and gyroscopes are introduced in the following subsections, while INS solution propogation and INS errors are introduced in the next two subsections respectively.

2.2.1 Accelerometers

A typical accelerometer contains a proof mass that is free to move along the sensitive axis of the accelerometer with respect to the accelerometer case. A simple accelerometer can be seen in Figure 2.2.

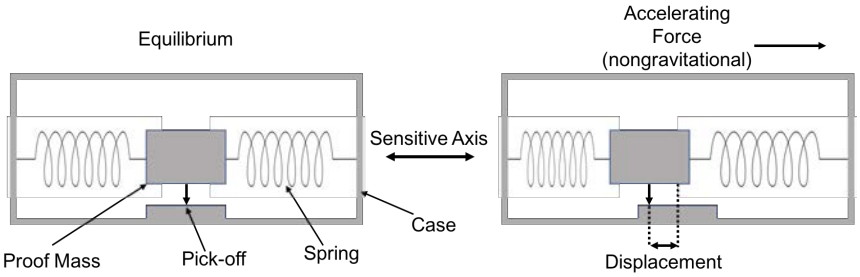

Figure 2.2: A Simplified Accelerometer [25].

The proof mass is restrained by a suspension which can be springs, a torquer such as an electromagnet, or vibrating beams. When an accelerating force is applied to the accelerometer case, the proof mass will initially continue at its previous velocity. The accelerometer case will move with respect to the proof mass and the extent of the movement is measured by a pickoff. In the case of an accelerometer using springs as the suspension, one spring will be compresssed, while the other spring is stretched. The stretching and compressing of the springs changes the amount of force transmitted from the springs to the proof mass. The case

will continue to move with respect to the proof mass until the acceleration of the proof mass due to the forces exerted by the springs matches the acceleration of the case due to the externally applied force. The resulting position of the proof mass relative to the case is proportional to the acceleration applied to the case. Using the measurement of the displacement of the proof mass with the pickoff allows for an acceleration measurement to be obtained. However, this does not apply to the gravitational force, which acts on the proof mass directly and applies the same acceleration to all components of the accelerometer equally. Therefore, there is no relative motion of the proof mass with respect to the case due to gravitational force. Because of this fact, accelerometers actually sense specific force, which is the nongravitational acceleration instead of the total acceleration [25]. The accelerometers in the KVH 1725 IMU used for this work are MicroElectroMechanical Systems (MEMS) accelerometers. MEMS accelerometers are small and light, quartz and silicon sensors which can be mass produced at a low cost using etching techniques with multiple sensors on a single wafer. These sensors can be built with the sensitive axis on the plane of the device or perpendicular to the plane of the device allowing all three accelerometers and their associated electronics to be built onto a single silicon chip [25]. More detail on the principles and different types of accelerometers can be found in [25].

2.2.2 Gyroscopes

The three main types of gyroscope, or gyro for short, widely used today are optical, vibratory, and spinning-mass. The KVH 1725 IMU used in this thesis uses an optical gyroscope, more specifically a Fiber-Optic Gyroscope (FOG). Figure 2.3 shows the main elements of a FOG. Optical gyroscopes operate on the principle that light travels at a constant speed in a given medium. A light is sent in both directions around a closed loop waveguide made of mirrors or optical fiber. If the waveguide is not rotating about an axis perpendicular to its plane, the path length is the same for both beams. However, if the waveguide is rotating about an axis perpendicular to its plane, the light travelling against the rotation will experience a shorter path delay than the light travelling with the rotation. By measuring the path delay for both beams of light, the angular rate of the waveguide with respect to inertial space can be determined [25].

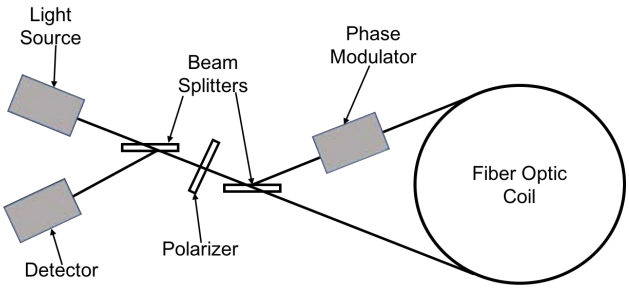

Figure 2.3: A Simplified Fiber Optic Gyroscope [25].

Vibratory gyroscopes contain an element such as a string or beam that is driven to undergo a simple harmonic motion. Other elements used are a pair of beams, tuning fork, ring, cylinder, or hemisphere. No matter the element used, the operating principle of the gyroscope is to detect the Coriolis acceleration of the vibrating element when the gyro is rotated. As the gyroscope rotates, the Coriolis acceleration instigates a simple harmonic motion along the axis perpendicular to the driven vibration and the projection of the angular rate vector. The amplitude of the harmonic motion is proportional to the angular rate. The motion of the vibrating element is constrained along one of the axes perpendicular to the driven vibration, so only a rotation about the input axis yields a significant oscillation in the output axis. The output vibration is detected by the gyroscope and then converted into a rotation rate. Most vibratory gyroscopes are low-cost and low-performance and typically employ MEMS technology [25, 34].

The third major type of gyroscope is the spinning-mass gyroscope. Spinning mass gryos use the the principles of conservation of momentum to detect rotation. A motor spins a mass about one axis. If a torque is then applied about an axis perpendicular to the spin axis, the mass rotates about the axis perpendicular to both the spin axis and the applied torque [25], allowing measurement of the angle of rotation.

2.2.3 Inertial Measurement Units

An Inertial Measurement Unit (IMU) is mainly comprised of accelerometers, gyroscopes, a processor, storage for calibration parameters, a temperature sensor, and power supplies. Most IMUs have three accelerometers and three single degree of freedom gyroscopes all mounted so the sensitive axes are orthogonal. IMUs with fewer than six sensors are called partial IMUs

and are sometimes used for land navigation [25]. The KVH 1725 IMU used for this work is a full six degree of freedom IMU.

The IMU's processor performs unit conversions, compensates for known error sources, and performs checks for sensor failure. The processor converts the sensor outputs into specific force and rotational rates. In some IMUs, these measurements are integrated over the sampling interval yielding delta values. Gyroscope deltas are attitude increments, but accelerometer deltas are not velocity increments due to the fact accelerometer measurements are specific force instead of acceleration [25]. Output rates for IMUs typically vary between 100 and 1,000 Hz. The IMU used for this work is sampled at 100 Hz.

The inertial sensors in an IMU exhibit constant errors that can be calibrated in a laboratory and stored in memory enabling the IMU processor to correct these errors. The calibration parameters generally include accelerometer and gyroscope biases, scale factor and cross-coupling errors, and g-dependent biases. These errors also vary with temperature, so the calibration is performed at a wide range of temperatures and the IMU processor uses the on-board temperature sensor to correct the errors at the internal temperature of the IMU. For low cost sensors, the same calibration coefficients may be applied to a whole production batch of sensors; however, to fully capture the cross-coupling errors, a per IMU calibration must be performed. Another source of error the IMU processor can correct for is known as the size effect. In order to compute a navigation solution for a single point in space, the IMU measurements must also apply to a single reference point. However, the size of inertial sensors dictate placement up to a few centimeters apart. This spacing presents no issue for gyroscopes, but can induce error for accelerometers. An accelerometer rotating around the reference point will sense a centrifugal acceleration that is not observed at the reference point. Angular acceleration around the reference points will cause the accelerometer to sense a Euler force [25].

2.2.4 INS Propagation

An INS is a dead-reckoning navigation unit which employs measurements from inertial sensors such as accelerometers and gyroscopes to propagate a position and attitude solution from an initial location. The position solution is maintained by integrating velocity, which is

maintained by integrating acceleration measurements from an IMU. The attitude solution is maintained by integrating angular rate measurements from the IMU. After initialization, the INS typically propagates the solutions forward without any information from the environment [25].

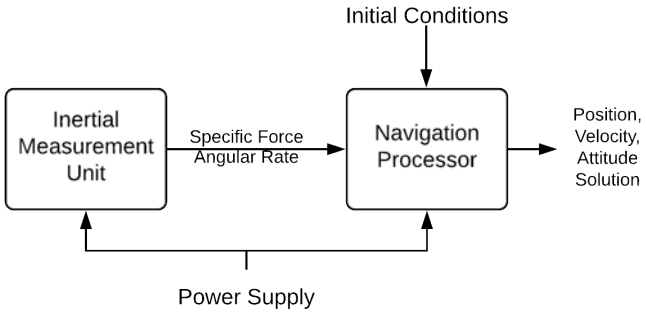

Figure 2.4: Basic Schematic of an Inertial Navigation System [25].

As shown in Figure 2.4, and INS is made up of an IMU and a navigation processor. The IMU measures specific forces and angular rates, which the navigation processor then uses to compute a change in position and attitude solution. The navigation processor may be packaged with the IMU, or may be implemented separately, but the function is the same in either case. The term inertial navigation system is applied to all architectures in which a three dimensional navigation solution is obtained from inertial measurements. The basic steps involved to find the navigation solution at each epoch are as follows [25]:

1. Update attitude solution using angular rate measurements.

2. Transform IMU body frame to the navigation frame.

3. Transform specific force measurements to accelerations and update velocity.

4. Update position solution using velocity solution.

Often, the INS is implemented in an integrated navigation system where estimates from the IMU and INS are corrected using outside measurements and estimates from the integration algorithm. An example of this is the GPS/INS algorithm discussed in Section 2.4.

2.2.5 INS Errors

All accelerometers and gyroscopes exhibit biases, scale factor and cross-coupling errors, and random noise, to some extent. Depending on the sensor type and quality, there may be higher-order errors. The IMU model used in this work only models errors due to random noise and bias. The model assumes the temperature-dependent variation of each error source is corrected by the IMU processor.

Each systematic error source has four components consisting of a fixed contribution, a temperature-dependent contribution, a run-to-run variation, and an in-run variation. The fixed contribution is present each time the sensor is used and corrected by the laboratory calibration data. The temperature-dependent component may also be corrected by the laboratory calibration data. When the temperature-dependent variation is not corrected, the systematic errors exhibited by the sensor will exhibit some variation during the first few minutes of operation while the sensor is warming up to normal operating temperature.

The run-to-run variation of each error source is a bias which s different each time the sensor is used, therefore, it cannot be corrected by laboratory data. However, the run-to-run variation remains constant within any given run, thus can be corrected by INS alignment or integration algorithms, which are described in more detail in [25]. The in-run variation of the error source slowly changes throughout the course of a run and cannot be corrected by the IMU or an alignment process. The in-run variation may be corrected through integration with other navigation sensors but is difficult to observe in practice [25]. Additionally, sudden step changes in errors may occur if the IMU is subjected to a large shock such as a launch from a gun [33].

Biases are constant errors exhibited by all accelerometers and gyroscopes and are independent of specific force and angular rate. In most cases, biases are the dominant error source in inertial sensors and is sometimes called the g-independent bias. Biases are typically split into static and dynamic components, with the static component representing the fixed bias. The fixed bias may also be known as turn-on bias or bias repeatability and is comprised of the run-to-run variation plus the residual fixed bias remaining after sensor calibration. The static

component is constant throughout the operating period, but varies from run to run. The dynamic component of the bias is also known as the in-run bias variation or bias instability. The bias instability typically varies over periods in the order of a minute or so and incorporates the remaining temperature-dependent bias after sensor calibration. The dynamic portion of the bias is typically around 10% of the static bias [25]. Table 2.2 shows typical accelerometer and gyroscope biases for the various grades of IMUs.

Table 2.2: Typical Accelerometer and Gyro Biases for Different Grades of IMU [25]

IMU Grade	Accelerometer Bias (m s^{-2})	Gyro Bias ($^\circ$ hr^{-1})
Marine	10^{-4}	0.001
Aviation	3×10^{-4}-10^{-3}	0.01
Intermediate	10^{-3}-10^{-2}	0.1
Tactical	0.01-0.1	1-100
Consumer	$>$0.03	$>$100

Scale factor error represents the difference of the input-output gradient of the sensor from unity following conversion by the IMU. The accelerometer scale factor error is proportional to the true specific force along the sensitive axis while the gyro scale factor error is proportional to the true angular rate about the sensitive axis. Cross-coupling errors arise from misalignment of the sensitive axes of the inertial sensors with respect to the orthogonal axes of the body frame of the IMU. Cross-coupling errors are due to manufacturing limitations and are sometimes referred to as misalignment errors. These errors cause the sensor to be sensitive to specific force or rotational velocities along axes orthogonal to the actual sensitive axis. Axis misalignment also causes additional scale factor errors, but these are typically two to four times smaller than the cross-coupling errors. For consumer-grade MEMS sensors, the cross coupling errors of the sensor itself can exceed the errors due to mounting misalignment [25].

Random noise is another main error source present for all inertial sensors and is due to a number of sources. Electrical noise limits the resolution of the sensors especially in MEMS sensors where the signal is very weak. Pendulous accelerometers exhibit noise due to mechanical instabilities while lock-in effects of an RLG manifest as noise. The spectrum of noise for

frequencies below 1 Hz is approximately white. MEMS sensors can also exhibit significant high-frequency noise [18]. This noise averages out over the order of a second within the IMU body frame so passing the sensor outputs through navigation equations will mitigate most of the effects of the high-frequency noise. However, in high-dynamic situations, this noise will not average out fully in the frame used to calculate the navigation solution, so caution must be exercised when choosing sensors. Many manufacturers quote the standard deviation of the total noise, which includes white and high frequency, at the sensor output. Another source of noise in inertial sensors is the quantization of the IMU outputs due to digitizing the sensor output. The sensor output is rounded to an integer multiple of a constant, which is known as the quantization level [25]. Word lengths of 16 bits are typically used on tactical grade sensors, yielding quantization levels on the order of 10^{-4} m/s and 2×10^{-6} rad when integrating IMU measurements. Consumer-grade sensors typically use a shorter word length of 8 to 12 bits, so quantization errors are higher at 10^{-3} m/s and 2×10^{-5} rad. The quantization level is usually slightly less than the quoted noise standard deviation. Quantization errors have a linerar distribution, but with sufficient samples, average to have a Gaussian distribution due to the central limit theorem. More information on the described IMU errors and further error sources can be found in [25].

2.3 Odometry

Odometry is another commonly used technology for ground vehicle navigation and is the determination of a vehicle's speed and distance traveled by measuring the rotation of the vehicle's wheels through the use of an odometer. The odometer has traditionally been mounted to the vehicle's transmission shaft, but newer vehicles have a sensor on each wheel. These sensors are known as Wheel Speed Sensors (WSS) and are used for Antilock Braking Systems (ABS). Robots also commonly use wheel speed sensors for odometry measurements [25]. Because these sensors need to last for the lifetime of the vehicle, noncontact sensors known as rotary encoders are used. Most of the rotary encoders used employ a ferrous wheel mounted on the transmission shaft or wheel axle. As each tooth passes a sensor, the magnetic flux density varies producing a pulsed electrical signal when measured. As a result, these sensors are commonly

referred to wheel pulse or wheel tick sensors. The number of pulses measured is proportional to distance traveled. Differentiating the pulses with respect to time yields the vehicle's ground speed. Low cost sensors are typically passive and use the principle of variable reluctance, but exhibit poor signal-to-noise levels and thus are vulnerable to vibration and interference. These sensors do not work well at speeds below about 1 m/s and are not recommended for navigation. Active sensors sometimes used are based on the Hall effect and yield a stronger signal but are more expensive [25, 62, 28]. Platforms in which higher resolution is required and where dirt is not an issue may employ optical encoders. Wheel speed measurements for road vehicles can typically be accessed though the Controller Area Network (CAN) but often contain large quantization errors. Differentiating the wheel rotation, normally expressed as a pulse count, typically yields higher-precision wheel speed measurements [59]. The WSS used for this work are optical encoders mounted to the drive wheels of the vehicle. Errors present on wheel speed sensors include wheel slip, incorrect estimation of wheel radius, and quatization error [57, 26].

2.3.1 Linear Odometry

Many navigation algorithms use wheel speed measurements for linear odometry, the equations for which can be found in [25]. However, relying only on WSS for velocity and distance estimation can prove troublesome. WSS measure the distance traveled over the ground, not the horizontal distance traveled. Thus anytime the vehicle travels on a sloped surface, the horizontal distance traveled will be overestimated; however, vehicle roll and road banking do not affect measurements. For slopes up to 8 degrees, the error in measurements will be less than 1% [25]. If the pitch of the surface is known using an IMU or GNSS velocity measurements, the speed and distance measurements may be corrected.

The dominant error source in linear odometry is scale factor error due to incorerct estimation of the wheel radius. Tire wear over the life of the tire can change the effective raduis by up to 3% [25] while changes in temperature, pressure, load, and speed can change the wheel radius on the order of 1% [8, 19, 54]. As a result, calibrating WSS using external navigation sensors such as GNSS is common practice in situations where linear odometry is used to aid in computing a navigation solution [25]. Because the ferrous wheels or optical encoders used have

a limited resolution, quantization may also be a significant source of error for short-term velocity measurements. However, the long term position error resulting from this quantization is negligible due to the fact quantization errors are always corrected by subsequent measurements [9]. Road surface unevenness also results in random errors.

Wheel slip due to rapid acceleration or braking cause WSS to produce false measurements of vehicle velocity [7]. Many navigation algorithms detect and filter out these false measurements using integrity monitoring techniques as used in [25]. ABS and traction control systems detect wheel slip by comparing WSS measurements to accelerometer measurements. Drive wheels are subject to more slippage, so odometry using nondriving wheels is typically more reliable [54]; however, if the vehicle is rear-wheel drive and front-wheel steer, steer angle measurements are required for odometry using the non-driving wheels.

2.3.2 Differential Odometry

Differential odometry may be employed when individual wheel speed measurements are available. The vehicle's yaw rate may be calculated by Equation (2.9)

$$\dot{\Psi} = \frac{v_{rL} - v_{rR}}{T_r} \tag{2.9}$$

from rear wheel velocities, or by Equation (2.10)

$$\dot{\Psi} = \frac{v_{fL} - v_{fR}}{T_f \cos\delta_f} - \dot{\delta}_f \tag{2.10}$$

from the front wheels. Velocity at each wheel is denoted by v, where subscripts r and f denote rear and front wheels respectively. Subscripts L and R specify left and right wheels respectively, T is the track width of the vehicle, and δ_f is the steer angle of the front wheels. The heading of the vehicle may be updated by integrating the resulting yaw rate measurements over the sampling period of the sensors.

Errors arise in differential odometry due to sloped, banked, or uneven terrain but not by vehicle roll. When the heading change is small, slope effects may be corrected by dividing

the yaw rate or heading change by the cosine of the terrain slope. Correcting odometry yaw rate measurements on banked terrain must be done through integration with other navigation sensors when roll and pitch are unknown [25]. Scale factor errors also pose a large problem to differential odometry. A 1% difference in scale factor error between the left and right wheels yields a yaw rate error around 3 deg/s at 10 m/s [25]. Scale factor errors in differential odometry are affected by errors in track widths and wheel radius, both of which may change each time the tires on the vehicle are replaced [30]. Therefore, like in linear odometry, calibration of scale factor error using external navigation sensors is common practice [25].

Road surface variations are a large source of error for differential odometry, much more so than linear odometry. A pot hole or bump which affects only one side of the vehicle may produce a 1 degree heading error while only producing a 1.5 cm error in distance traveled [25]. Road camber also introduces error in scale factor during turns and changes in camber on straight portions of terrain produce false turn measurements [60]. Curved road camber may also bias differential odometry by changing effective tire radii, but may be corrected using vehicle roll measurements from an accelerometer [59]. Like linear odometry, differential odometry is affected by wheel slip and WSS quantization with the latter being negligible in long term yaw measurement.

2.4 GPS/INS Algorithm Overview and Variations

As introduced in Section 2.2, inertial navigation provides a means of obtaining a position and attitude solution from an initial condition. An advantage of INS is position and attitude solutions that are provided at a high rate of at least 50 Hz with low short term noise. However, disadvantages include the INS must be initialized and accuracy degradation over time due to the fact the inertial measurement errors are integrated in the navigation equations. INS capable of providing effective navigation for more than a few minutes are also very expensive, at around $100,000 [25].

Alternatively, GPS provides high long-term position accuracy when compared to INS, with errors limited to a few meters for standalone GPS. The cost for a GPS sensor is much lower as well, with equipment available from $100. The drawback to using GPS for positioning

is the low output rate (typically less than 10 Hz), the short term noise is high, and standard GPS equipment does not provide an attitude solution. GPS is also vulnerable to environmental effects and obstructions and therefore cannot be relied on for a continuous navigation solution [25]. This is especially true for autonomous vehicles which require a position and attitude solution at a high rate to use in vehicle control.

The advantages and disadvantages of INS and GPS are complimentary so by using them together, the advantages of both systems may be combined to give a high output rate position solution with low short and long term noise that is resistant to environmental effects. The GPS solution prevents the INS solution from drifting and the INS solution smooths the GPS solution and mitigates signal outages [25]. However, since the GPS error drift has a time constant of around 30 to 60 minutes, the IMU measurements cannot filter the large, slowly varying GPS errors. The combination of GPS and INS is typically realized through state estimation algorithms, the base of which is the Kalman filter [25]. The Kalman filter is designed only for linear systems, therefore an Extended Kalman Filter (EKF) is employed in many cases including the GPS/INS algorithm found in the Auburn University GPS and Vehicle Dynamics Laboratory (GAVLab) navigation unit used in this thesis.

Several different algorithm architectures including loosely coupled, closely coupled, and deeply coupled, are discussed in [48, 25]. The loosely coupled architecture is the simplest version and uses the GPS position and velocity solution to compare to and correct the INS solution. The main drawback of this architecture is when there are less than four satellites visible, the GPS receiver cannot compute a solution and thus there are no measurements to correct the INS solution. On the other hand, the closely coupled architecture uses individual SV measurements to make use of information from the GPS receiver even when there are less than four satellites. The GAVLab navigation unit employs the closely coupled architecture, which can be visualized in Figure 2.5.

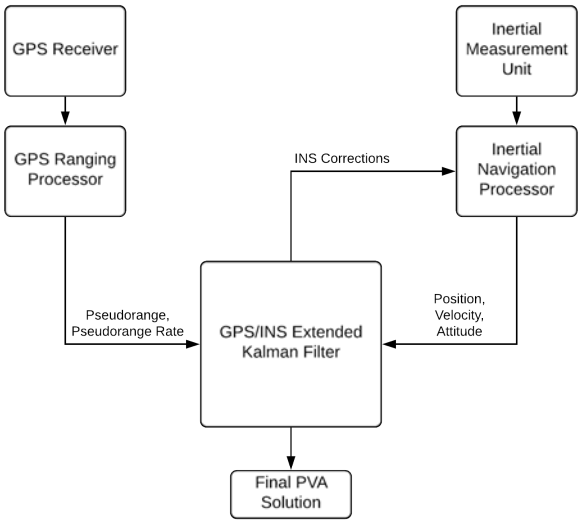

Figure 2.5: Closely Coupled GPS/INS Integration Architecture [48].

The filter is based on the INS solution and predicts pseudoranges and pseudorange rates for each satellite in view. The predicted pseudoranges and rates are then compared to measurements from the GPS receiver and the error is used to correct the navigation solution. The closely coupled and subsequent deeply coupled architectures cannot be implemented with low cost receivers due to the fact the receivers typically do not output the required measurements [48]. The deeply coupled architecture allows the navigation algorithm to use GPS aiding in scenarios with degraded signals, achieved by feeding corrections from the navigation filter back into the GPS receiver at the cost of increased complexity.

2.5 GAVLab Navigation System

As stated, the goal of this work is to develop a HIL/SIL system for easy testing of navigation units. The specific navigation system used for testing in this thesis was built by Auburn University's GAVLab; however, the designed system will work with any navigation unit, so long as the algorithm is implemented in the Robotic Operating System (ROS) framework and can accept outside signals. The navigation algorithm in the GAVLab unit is implemented on a Linux PC allowing algorithms to be readily interchangeable, so long as they are implemented using the ROS framework. The algorithm subscribes to GPS, IMU, and WSS topics published

by the respective sensor drivers, and uses them to compute the navigation solution. A simplified representation of a navigation unit can be seen in Figure 2.6. The GAVLab navigation unit and Linux PC are shown in Figures 2.7 and 2.8.

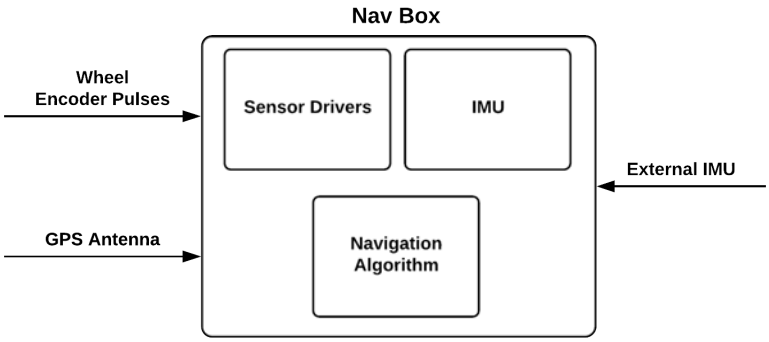

Figure 2.6: Simplified Navigation Unit.

Figure 2.7: GAVLab Navigation Unit.

Figure 2.8: Advantech Linux PC.

The sensors employed by the GAVLab navigation unit include a KVH 1725 IMU, wheel encoders, and Novatel OEM 628 GNSS receiver. Each sensor is connected to a Linux PC and sampled using C++ code written by the GAVLab or open source code, which are wrapped in ROS. All sensors are sampled using code in the ROS environment so that outputs are readily available to any other node and so they have a common timing source. The KVH 1725 outputs RS-422 serial signal that is converted using a RS-422 to USB adapter due to the fact the Linux PC does not have RS-422 serial ports. The wheel encoders used by the GAVLab navigation unit are optical incremental encoders mounted on the rear wheels of the test vehicle as shown in Section 4.1.1. The encoders have a resolution of 1,000 counts per revolution and output a 5 volt quadrature signal. The encoder output signals are sampled by a pair of US Digital QSB-D USB encoder readers that are polled at 100 Hz by the Linux PC which output the total number of encoder pulses counted.

Chapter 3

Development of Simulation Environment

3.1 System Architecture

The work presented in this thesis is developed with a two-fold system architecture. First, users can choose to run the system in a software only mode, known as Simulation In the Loop (SIL). In SIL mode, all signals from the various sensors are simulated, including GPS, IMU, WSS. In the second mode of operation, users can incorporate Hardware In the Loop (HIL) modules in order to integrate and test hardware in the simulation. The hardware additions to the system include a Spectracom real-time GPS Radio Frequency (RF) signal generator, analog encoder pulses generated using an Arduino Due for WSS, and a serial interface for IMU measurements to be fed into a navigation unit. The navigation unit used for testing and validation is the Auburn navigation unit described in Section 2.5. The overall system architecture with hardware additions included can be seen in Figure 3.1.

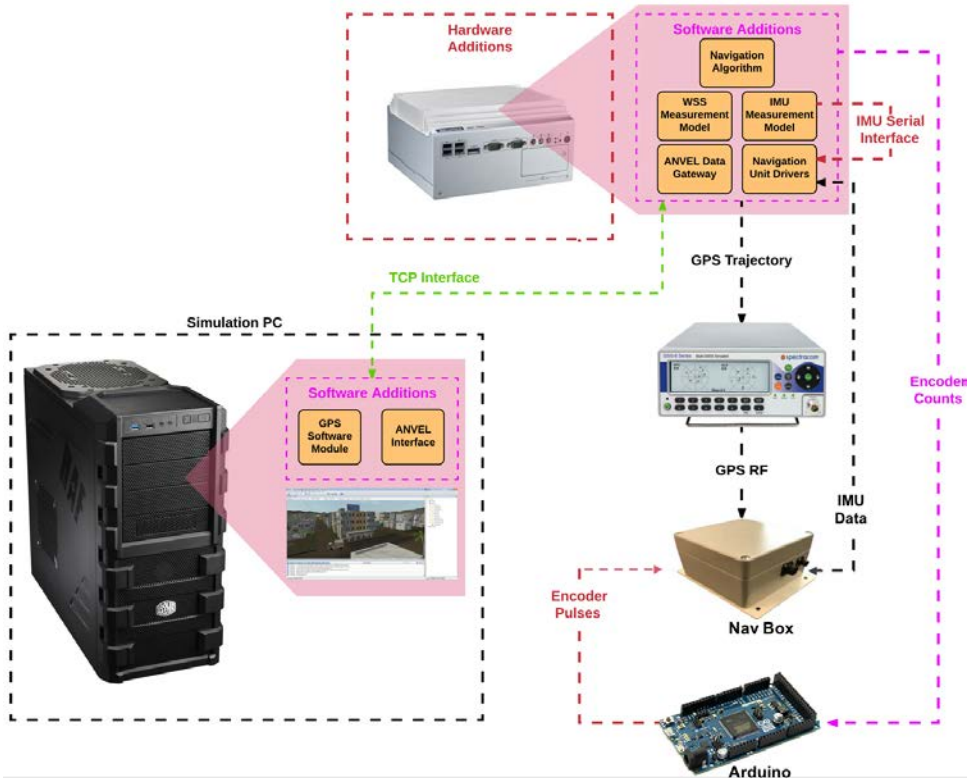

Figure 3.1: System Architecture Overview.

Each sensor system including GPS, IMU, and WSS has a respective software and hardware module, and the system is designed with a modular framework such that only the desired modules are used at any given time. If the user desires only IMU data, either the IMU software or hardware module may be used. If the user desires data from all sensors modules, the system may be operated with all modules used at once.

The vehicle simulation is run in Autonomous Navigation Virtual Environment Laboratory (ANVEL), a rapid prototyping tool developed for the U.S. Army Engineer Research and Development Center (ERDC). ANVEL specializes in Unmanned Ground-Vehicle (UGV) technologies and is used as a virtual proving ground for the development of new UGV technologies. The Simulation Personal Computer (SimPC), seen in Figure 3.1 is only used for the ANVEL simulation, ANVEL interface, and GPS simulation. The ANVEL interface allows the system to import and export measurements to and from the simulation environment. The SimPC is connected via Transmission Control Protocol (TCP) interface with a Linux PC, which houses the IMU and WSS modules.

All of the software and hardware modules developed for this work employ the ROS framework due to the fact it is widely popular in industry and military applications. ROS provides standard messages known as ROS topics that can be easily published by or subscribed to in each of the developed modules from the ROS database, allowing easy communication between modules and a common time-stamping system.

The developed modules are designed to provide simulated sensor output that replicates actual sensor outputs as closely as possible. If the user desires the HIL system implementation to incorporate hardware measurement errors, the Spectracom simulator, encoder pulse generation, IMU serial interface, and navigation unit may be implemented as seen in Figure 3.1. The software is also developed such that the navigation unit may be placed on a rate table to incorporate real IMU measurements, but rate table implementation is outside of the scope of this work. The development of each of the previously mentioned modules, as well as the ANVEL interface, is discussed in the following sections.

3.2 ANVEL Interface

The advantage of using ANVEL is its open software framework. This framework allows researchers the ability to readily extend ANVEL to meet any needs through the use of the external Application Program Interface (API) and plugins. The external API gives researchers a set of commands and functions that may be used to query vehicle or sensor information. Plugins allow researchers to extend ANVEL by adding new sensors, vehicles, VTI models, or entirely new behaviors. ANVEL's specialities make it a perfect development bed for the work presented in this thesis. The availability of a perfect ground truth from ANVEL enables easy debugging during system development and accurate error analysis when testing navigation and control algorithms.

The first step in implementing the HIL/SIL with ANVEL is developing an interface between ANVEL and the rest of the system for the transmission of simulation data. This interface and data transmission are handled via a plugin for ANVEL. The vehicle state information plugin developed for this work was provided by the U.S. Army Tank Automotive Research Development and Engineering Center (TARDEC) and gathers vehicle state information and

sensor measurements from ANVEL in real time and makes them available for use by developers as required. Vehicle state information and simulated, ideal sensor output from ANVEL are published by the SimPC communication plugin via TCP connection. The data is received and parsed by a ROS node on the Linux PC and published to a ROS database via standard and custom ROS topics. The GPS software module, further described in Section 3.3, is also written as a plugin to ANVEL. The GPS software module runs on the SimPC and is controlled by ANVEL. Data from the GPS software module is also published via TCP and read by a separate ROS node on the Linux PC, which then publishes the GPS messages to the database.

In certain test situations, commanding vehicle position in ANVEL may be necessary. One such situation is testing of the positioning and ray tracing of the GPS modules against experimental data. In order to facilitate position commands to ANVEL, a ROS node is developed which takes a text file of user positions and employs the ANVEL external API to command ANVEL to place the vehicle in the desired location. Commanding positions in ANVEL in this manner does not allow ANVEL to simulate the physics of the vehicle, therefore vehicle velocity and acceleration outputs from ANVEL are invalid.

3.3 GPS Software Module

The GPS software module outputs all GPS measurements used by common GPS/INS algorithms. Outputs include pseudorange, carrier phase, Doppler shift, and a final PVT solution and are designed to be representative of outputs from the Novatel receiver in the GAVLAB navigation unit. The basic flow of the GPS software module is shown in Figure 3.2.

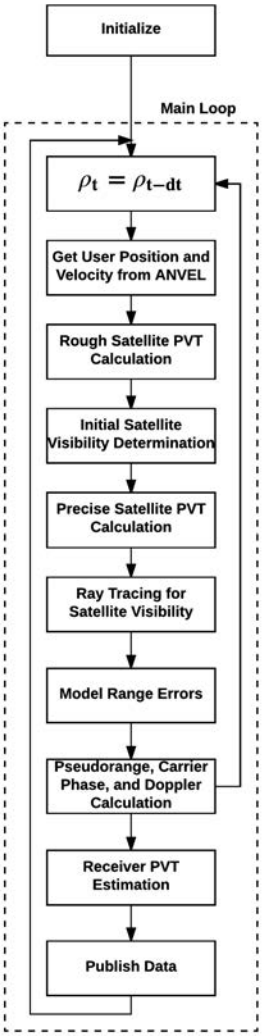

Figure 3.2: Flow of GPS Software Module.

All simulated outputs are generated using the perfect user location known from ANVEL and satellite positions simulated using ephemeris data from the CDDIS [11]. The entire GPS software module is built as a plugin to ANVEL in order to make use of the ray collision methods implemented in ANVEL. Building the GPS software module as a plugin makes the perfect ANVEL position immediately available to the module and the calculated satellite positions do not need to be returned to ANVEL to test for ray collisions. As with the vehicle information plugin, the GPS software module plugin publishes the simulated GPS measurements and satellite availability information via TCP to be parsed by a gateway node on the Linux PC. The measurements are then published to the ROS database by the gateway node. Before the module can begin the simulation, initialization steps must be taken, pictured in Figure 3.3.

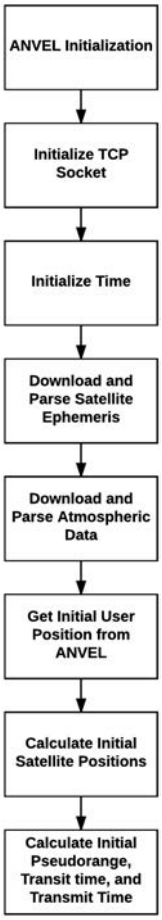

Figure 3.3: GPS Software Module Initialization Steps.

The first step in the initialization stage is getting pointers to the physics objects representing the GPS sensor from ANVEL. Having the pointers allows the GPS simulation to query the physics simulation for true sensor position and velocity. Next, simulation time is initialized to the desired preset time. The time can be initialized to any time up to the current time since ephemeris data is not available for future dates. The module uses the current time to download the appropriate ephemeris and atmospheric data from [11, 12, 2], which is then parsed into matrices to be used in the main simulation loop. The atmopsheric data downloaded from [12] is a grid containing Total Electron Content (TEC) data for varying latitudes and longitudes, while the data downloaded from [2] contains four grids representing the different parameters for the tropospheric model used. The atmospheric data from [12, 2] is used to estimate the delays actually experienced by the satellite signals. Details on the atmospheric delay calculations using this downloaded data can be seen in Section 3.3.2. Next, initial satellite positions

and velocities for all 32 GPS satellites are determined using the ephemeris data and the initial pseudoranges are calculated using the satellite positions along with the known user position from ANVEL. The initial pseudoranges are then used to calculate the initial signal transit times and thus transmit times, ending the initialization process.

The main simulation is entered once the initialization steps are complete. As can be seen in Figure 3.2, the first step after carrying over pseudoranges is to determine the user's true position and velocity from ANVEL. Retrieving the user position and velocity only requires querying the underlying ANVEL physics simulation for the desired quantities. Next, the simulation calculates rough satellite positions at the time of signal transmission using the previous pseudoranges to calculate signal transit time. Initial satellite visibility for the current time step is then determined, consisting of only satellites over the mask angle. The mask angle is the satellite's minimum elevation angle to be used by the receiver. Once the satellites over the mask angle are determined, the simulation calculates more exact satellite positions in an iterative process. The second satellite visibility check is then performed by conducting ray tracing between the user position and all satellites over the mask angle. Blocked satellites are recorded and no longer used in the simulation for the current time step. The measurement generation that follows is only performed for the satellites currently in view of the receiver, saving computation time. For satellites that are not in view for the current time step, measurements are propagated using the previous measurements. The next step in the loop is the calculation of atmospheric errors, both estimated and those calculated by the receiver's atmospheric models. Next, the code-based and carrier based pseudoranges are calculated, with calculation of the Doppler shift for each visible satellite following. The final step for the current simulation step is to use the satellite positions, velocities, Doppler shift, and pseudoranges to calculate a PVT solution. At this point, all desired measurements have been generated and are packaged and sent over TCP to the additional PC running the ROS environment, which receives the measurements and publishes them as ROS topics. All calculations referenced are outlined in the following sections.

3.3.1 Satellite Position and Velocity

Satellite positions are calculated using the quasi-Keplerian orbital elements from the broadcast ephemeris data. In order to calculate the actual position of the satellite at the signal transmit time, the signal transit time for each satellite must be known. Each satellite is a different distance from the receiver, therefore, transit time for each satellite will be different. The satellite position and clock correction calculation is performed using the downloaded ephemeris and is fully detailed in the GPS ICD [40]. Equations for satellite position determination are shown in Appenix A.

When calculating the rough satellite positions for initial visibility determination, the pseudoranges from the previous time step are used to calculate transit time. Because the rough position is used to determine which satellites are over the horizon, the rough positions and subsequent pseudoranges are calculated for all satellites. When calculating exact satellite positions for the visible satellites, an iterative process is used to calculate transit time and satellite positions due to the fact they are correlated. First, the position of the satellite is calculated using the previous pseudorange for that satellite, or an arbitrary pseudorange during initialization. The range between the receiver and the satellite is then calculated using Equation (3.1).

$$r_k = \sqrt{(x_{s_k} - x_u)^2 + (y_{s_k} - y_u)^2 + (z_{s_k} - z_u)^2} + r_{e_k} \tag{3.1}$$

where x, y and z represent the coordinates of the satellite and user in ECEF with subscript s_k denoting satellite k and subscript u denoting the user. The range due to the rotation of the earth during signal transit for each satellite is represented by r_{e_k}, calculated using Equation (3.2) below.

$$r_{e_k} = \frac{w_e}{c}(x_k y_u - y_k x_u) \tag{3.2}$$

The rotation rate of the earth is represented by w_e, $7.2921159 \times 10^{-5} rad/s$, and c is the speed of light, 299,792,458 m/s. The transit time is then calculated by Equation (3.3).

$$t_{tr_k} = \frac{r}{c} \tag{3.3}$$

The corresponding transmit times t_{t_k} are calculated by Equation (3.4)

$$t_{t_k} = t - t_{tr_k} \tag{3.4}$$

where t is the current time. New satellite positions are then found using the new transmit time and a new range is calculated using Equation (3.1). The difference between the two ranges is calculated in Equation (3.5)

$$\Delta r_k = r_{k_2} - r_{k_1} \tag{3.5}$$

where r_{k_1} is the range calculated using the previous pseudorange and r_{k_2} is the range calculated using the new satellite positions. If the difference between the two ranges meets the predetermined tolerance, the satellite positions and transmit times calculated using r_{k_2} are considered final for the current time step. If the tolerance is not met, the newly calculated satellite positions are set as the previous satellite positions and the process is repeated until the tolerance is met.

3.3.2 Atmospheric Effects

To make the GPS simulation as accurate as possible, two types of errors are calculated for each layer of the atmosphere. The first type is the true atmospheric error, calculated using downloaded data from [12, 2], which represents the best estimate of the atmospheric delays the RF signal experienced while passing through the atmosphere. The second type is the atmospheric error the receiver calculates based on different atmospheric models which the receiver then attempts to use the to correct the measured pseudoranges. The GPS software module adds the estimated atmospheric errors to the pseudoranges and subsequently subtracts the receiver calculated errors to model what the receiver experiences, but only for the SIL module. Determination of the estimated and receiver calculated atmospheric errors are outlined below.

Estimated Ionosphere Error

The true ionospheric error experienced by the GPS signal is calculated using Total Electron Content (TEC) data downloaded from the CDDIS [12]. The steps outlined are first presented

in [36]. The first step in determining the true ionospheric error is to determine the Ionospheric Pierce Point (IPP), which is the point at which the signal from the SV passes through the ionosphere. The earth centered angle ψ is found from Equation (3.6)

$$\psi = \frac{0.0137}{E + 0.11} - 0.022 \tag{3.6}$$

where E is the satellite elevation angle in semicircles. Next, the IPP latitude is found using Equation (3.7)

$$\phi_I = \phi_u + \psi \cos A \tag{3.7}$$

with ϕ_u representing the user approximate geodetic latitude and A representing the azimuth angle of the satellite. If $\phi_I > 0.416$ then $\phi = 0.416$ and if $\phi_I < -0.416$ then $\phi_I = -0.416$. Now the longitude of the IPP is calculated using Equation (3.8)

$$\lambda_I = \lambda_u + \frac{\psi \sin A}{cos\phi_I} \tag{3.8}$$

where λ_u is the geodetic longitude of the user. The geocentric latitude of the IPP is then found using Equation (3.9)

$$\phi_c = \left[1 - e^2 \frac{R_N}{R_N + h}\right] \tan \phi_I \tag{3.9}$$

where e is eccentricity, defined as 0.0818191 for the World Geodetic Survey (WGS-84) [25]. The radius of curvature in the prime vertical R_N is given by Equation (3.10)

$$R_N = \frac{R_e}{\sqrt{1 - e^2 \sin^2 \phi}} \tag{3.10}$$

where R_e is the radius of the Earth, $6,378,137m$. Next, Equation (3.11) is used to convert GPS time to UTC time in order to find the correct TEC value from the downloaded data

$$t_{utc} = t_{GPS} - \delta t_{GPS/UTC} \tag{3.11}$$

where t_{GPS} is the current GPS time and $\delta t_{GPS/UTC}$ is the difference between UTC and GPS time, also known as leap seconds. The downloaded TEC data is now searched based on the time, latitude, and longitude to find the Vertical TEC (VTEC) value for the given satellite. The VTEC value must then be converted to a slant TEC using (3.12)

$$TEC = \frac{1}{\cos\zeta} \cdot VTEC \cdot 0.1 \tag{3.12}$$

with z denoting the zenith angle of the SV found using (3.13)

$$\zeta = \sin^{-1}\left(\frac{\sin\left(pi/2 - el\right)}{R_e + h_I} \cdot R_e\right) \tag{3.13}$$

where el is the SV elevation angle and h_I is the height of the IPP, $350,000$ m. The ionospheric delay is now calculated using the TEC value found in (3.14)

$$I_\rho(f) = \frac{40.3 \cdot TEC}{f^2} \tag{3.14}$$

with subscript ρ denoting pseudorange. The carrier phase ionospheric delay for the same frequency is simply the negative of the pseudorange delay. Note that Equation (3.14) is a function of frequency and can be used to find the ionospheric delay for any frequency signal.

Estimated Troposphere Error

The estimated tropospheric error is calculated using data downloaded from GGOS Atmosphere [2, 6]. The data is in the form of grids that are a function of latitude and longitude. A total of four grids are downloaded for each scenario, two for the zenith wet and dry delays, respectively, and two for the a coefficient for the wet and dry components of the delay. respectively. The mapping function for both the wet and dry components of the data is represented

by Equation (3.15)

$$mf_i(el) = \cfrac{1 + \cfrac{1 + a_i}{1 + \cfrac{b}{1 + c}}}{\sin(el) + \cfrac{a_i}{\sin(el) + \cfrac{b}{\sin(el) + c}}} \tag{3.15}$$

where subscript i denotes dry or wet, $b = 0.0029$, and c is found from Equation (3.16)

$$c = c_0 + \left[\left(\cos\left(\frac{doy - 28}{365} \cdot 2\pi + hem\right) + 1\right) \cdot \frac{c_{11}}{2} + c_{10}\right] \cdot (1 - \cos\phi) \tag{3.16}$$

where doy is the day of the year. Latitude is represented by ϕ and parameters b, c_0, c_{10}, c_{11}, and hem are specified in Table 3.1.

Table 3.1: Tropospheric Hydrostatic Mapping Function Parameters [6]

Hemsiphere	c_0	c_{10}	c_{11}	hem
Northern	0.062	0.001	0.005	0
Southern	0.062	0.002	0.007	π

The b and c of the wet mapping function are unchanged from the dry because the wet zenith delay is smaller than the dry by a factor of around 10, and the effect of variation of b and c is not significant [6]. The total tropospheric delay is then the sum of dry and wet zenith delays, T_{z_d} and T_{z_w}, multiplied by their respective mapping functions in Equation (3.17).

$$T_t = T_{z_d} \cdot mf_d + T_{z_w} \cdot mf_w \tag{3.17}$$

Receiver Calculated Ionosphere Error

The novatel receiver models atmospheric errors due to both the ionosphere and troposphere and uses these models to attempt to correct the measured pesudoranges. The receiver

uses the Klobuchar model for the ionosphere using parameters broadcast in the GPS navigation message. The ionosphere is somewhat volatile and therefore diffucult to accurately model; however, the use of the Klobuchar model reduces RMS range error by about 50 % [39]. The method used to calculate the receiver modeled ionospheric delay is presented first in [36], with a good translation found in [32]. The first step in determining the receiver ionosphere delay is calculating the geodetic latitude and longitude of the Ionospheric Pierce Point (IPP) using Equations eqs. (3.6) to (3.8). The geomagnetic latitude of the IPP is then found using Equation (3.18)

$$\phi_m = \phi_I + 0.064 \cos(\lambda_I - 1.617) \tag{3.18}$$

and the local time at the IPP is found from (3.19)

$$t_{IPP} = 43200\lambda_I + t \tag{3.19}$$

where t is the current GPS time. The time at the IPP is constrained $0 \leq t \leq 86,400$. Next, the amplitude and period of the ionospheric delay, A_I and P_I respectively, are calculated in Equations (3.20) and (3.21)

$$A_I = \sum_{n=0}^{3} \alpha_n \phi_m^n \tag{3.20}$$

$$P_I = \sum_{n=0}^{3} \beta_n \phi_m^n \tag{3.21}$$

where α and β are coefficients broadcast in the satellite navigation message. If $A_I < 0$, then $A_I = 0$ and if $P_I < 72,000$ then $P_I = 72,000$. The phase of the ionospheric delay is then calculated in Equation (3.22).

$$X_I = \frac{2\pi(t_{IPP} - 50400)}{P_I} \tag{3.22}$$

Next, the slant factor F is determined in Equation (3.23)

$$F = 1.0 + 16.0(0.53 - el)^3 \tag{3.23}$$

and finally the ionospheric delay can be calculated using Equation (3.24)

$$I_{L1_{GPS}} = \begin{cases} \left[5 \times 10^{-9} + \sum_{n=0}^{3} \left(1 - \frac{X_I^2}{2} + \frac{X_I^4}{24}\right)\right], & |X_I| \leq 1.57 \\ 5 \times 10^{-9}, & |X_I| \geq 1.57 \end{cases} \tag{3.24}$$

Note that (3.24) is the ionospheric delay in seconds for the GPS L1 frequency. The delay for GPS L2 or any other frequency signal may then be found using Equation (3.25)

$$I_f = \left(\frac{f_{L1_{GPS}}}{f}\right)^2 I_{L1_{GPS}} \tag{3.25}$$

where f is the frequency of the signal for which the delay is being found.

Receiver Calculated Troposphere Error

The method used to calculate receiver tropospheric correction is presented in [10] and has been adopted by Satellite-Based Augmentation Systems (SBAS) [46] with a good translation presented in [55]. As stated, the wet delay of the troposphere is volatile and not easily modeled, but the hydrostatic delay is stable and easily modeled allowing the receiver to calculate a reliable troposphere correction. The total tropospheric delay is a combination of the wet and dry delays shown in Equation (3.26)

$$T(el) = (T_{z,dry} + T_{z,wet})M(el) \tag{3.26}$$

where $T_{z,dry}$ and $T_{z,wet}$ are the zenith dry and wet tropospheric delays and $M(el)$ is the mapping function, also called an obliquity factor, which is a function of the elevation angle. The obliquity factor is given in Equation (3.27) [5]

$$M(el) = \frac{1.001}{\sqrt{0.002001 + \sin^2(el)}} \tag{3.27}$$

and is valid for any elevation angle over 5 degrees. The zenith wet and dry delays are calculated using meteorological parameters and the receiver height. The meteorological parameters

include pressure ($P(mbar)$), temperature ($T(K)$), water vapour pressure ($e(mbar)$), temperature lapse rate ($\beta(K/m)$), and water vapour lapse rate (λ_0). Each of the parameters used are calculated based on latitude (ϕ) and the day of the year (D) using (3.28)

$$\xi(\phi, D) = \xi_0(\phi) - \Delta\xi(\phi)\cos\left[\frac{2\pi(D - D_{min})}{365.25}\right] \tag{3.28}$$

where D_{min} is 28 for northern latitudes and 211 for southern latitudes. The $\xi_0(\phi)$ and $\Delta\xi(\phi)$ are the average and seasonal variation values for each parameter at the receiver latitude and are linearly interpolated from Tables 3.2 and 3.3.

Table 3.2: Average Meteorological Parameters [55]

Latitude (°)	Average				
	$P_0(mbar)$	$T_0(K)$	$e_0(mbar)$	$\beta_0(K/m)$	λ_0
15° or less	1013.25	299.65	26.31	6.30×10^{-3}	2.77
30°	1017.25	294.15	21.79	6.05×10^{-3}	3.15
45°	1015.75	283.15	11.66	5.58×10^{-3}	2.57
60°	1011.75	272.15	6.78	5.39×10^{-3}	1.81
75° or greater	1013.00	263.65	4.11	4.53×10^{-3}	1.55

Table 3.3: Seasonal Variation of Meteorological Parameters [55]

Latitude (°)	Average				
	$\Delta P(mbar)$	$\Delta T(K)$	$\Delta e(mbar)$	$\Delta\beta(K/m)$	$\Delta\lambda$
15°	0.00	0.00	0.00	0.00×10^{-3}	0.00
30°	-3.75	7.00	8.85	0.25×10^{-3}	0.33
45°	-2.25	11.00	7.24	0.32×10^{-3}	0.46
60°	-1.75	15.00	5.36	0.81×10^{-3}	0.74
75° or greater	-0.50	14.50	3.39	0.62×10^{-3}	0.30

The zenith delay terms $T_{z,dry}$ and $T_{z,wet}$ are calculated by Equations (3.29) and (3.30)

$$T_{z_0,dry} = \frac{10^{-6} k_1 R_d P}{g_m} \tag{3.29}$$

$$T_{z_0,wet} = \frac{10^{-6} k_2 R_d e}{(\lambda + 1) g_m - \beta R_d T} \tag{3.30}$$

where $k_1 = 77.604 K/mbar$, $k_2 = 382000 K^2/mbar$, $R_d = 287.054 J/Kg/K$, and $g_m = 9.784 m/s^2$. Next, the vertical delay at the receivers height (H) in meters are calculated using Equations (3.31) and (3.32)

$$T_{z,dry} = \left[1 - \frac{\beta H}{T}\right]^{\frac{g}{R_d \beta}} T_{z_0,dry} \tag{3.31}$$

$$T_{z,wet} = \left[1 - \frac{\beta H}{T}\right]^{\frac{(\lambda+1)g}{R_d \beta} - 1} T_{z_0,wet} \tag{3.32}$$

where $g = 9.80665 m/s^2$. Finally, Equation (3.26) is applied and the total receiver calculated ionospheric delay is found.

3.3.3 Satellite Visibility

A major advantage of the GPS software module is the ability to perform ray tracing in order to simulate satellites being blocked due to obstacles in the environment. This allows researchers the ability to test algorithms in environments with degraded GPS signals in simulation. The initial satellite visibility determination for each time epoch consists of determining which satellites are above the mask angle. The elevation angle in degrees for each SV is calculated using Equation (3.33)

$$el = \sin^{-1}\left(\frac{U_{s_k}}{r_{s_k}}\right) \tag{3.33}$$

with satellite positions represented in the East North Up (ENU) frame referenced from the current user position. The up component of the ENU position of each satellite is represented by U_{s_k} and the range between the user and satellite is represented by r_{s_k}. The satellite positions

are rotated from the ECEF frame into the ENU frame using Equation (3.34)

$$\begin{bmatrix} E_{s_k} \\ N_{s_k} \\ U_{s_k} \end{bmatrix} = \begin{bmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\sin\phi\cos\lambda & -\sin\phi\sin\lambda & \cos\phi \\ \cos\phi\cos l\lambda & \cos\phi\sin\lambda & \sin\phi \end{bmatrix} \begin{bmatrix} X_{s_k} - X_u \\ Y_{s_k} - Y_u \\ Z_{s_k} - Z_u \end{bmatrix} \tag{3.34}$$

where X,Y, and Z with subscripts s_k and u are the ECEF positions of the satellite and user respectively. The latitude and longitude of the user position are represented by ϕ and λ respectively. The purpose of the first satellite visibility check is to simply rule out satellites under the mask angle so that the simulation does not waste time calculating measurements for these satellites in the current epoch.

The second satellite visibility check is realized through the ray tracing functionality of ANVEL's built in physics functions that test for ray collisions. An example of the ray tracing capabilities of the GPS software module can be seen in Figure 3.4.

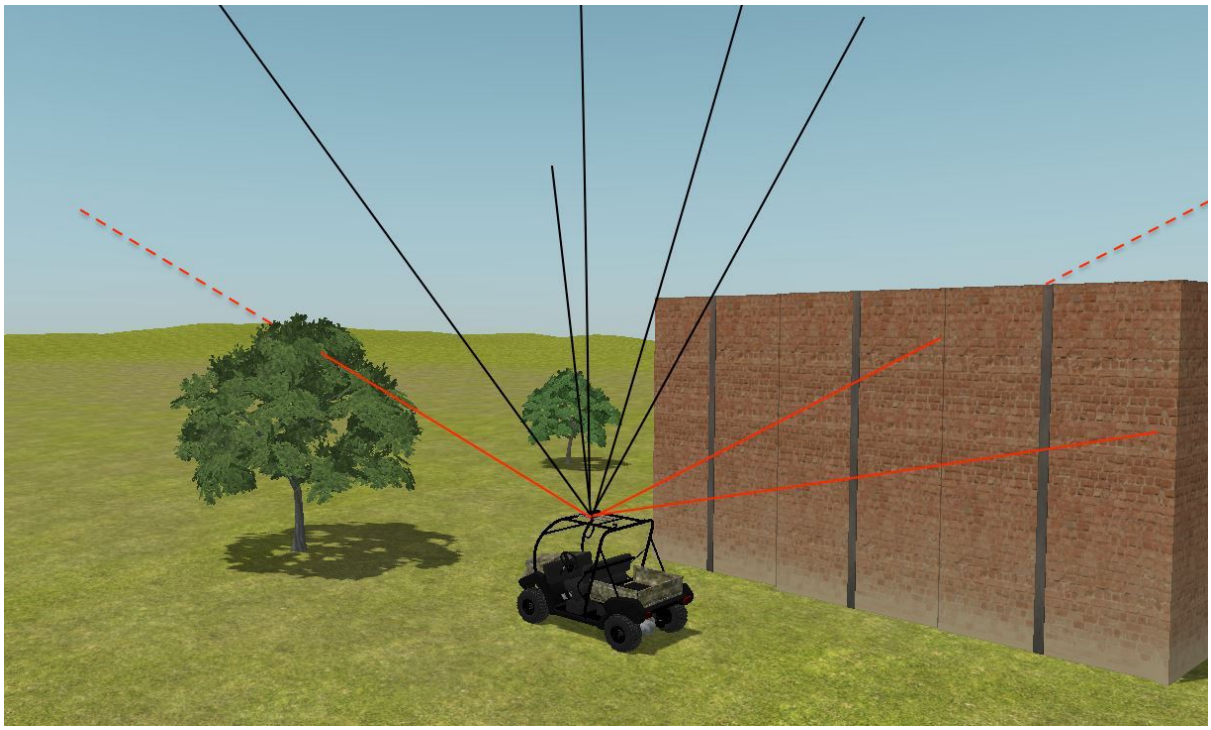

Figure 3.4: Visualization of Ray Tracing Capabilities in ANVEL.

The red lines represent a satellite that is blocked due to an obstacle, while the black lines represent satellites that are not blocked by any obstacles and therefore have line of sight to the antenna.

The ray tracing functions in ANVEL require an origin, ray direction, geometries to ignore, and the ray length. The origin is the local position of the sensor in the ANVEL coordinate system, the ray direction is the unit vectors from the sensor position to each satellite, and the ray length is set to 1,000 m so the function does not waste processing time testing for collisions 20,000 km into the sky. The geometries to ignore are the vehicle and the physics representation of the sensor. Ignoring the vehicle and sensor physics representations is necessary due to the fact the physics collision representations of the vehicle and sensor are simplified as boxes around their respective geometries. This causes the ray collision functions to return collisions with the vehicle the sensor is mounted on, even for satellites directly overhead with the sensor mounted to the roof of the vehicle. Obviously there is no true ray collision due to the vehicle in this case; therefore, ANVEL must be instructed to ignore the vehicle when computing ray collisions. Satellites that test positive for ray collisions are recorded and are not used for the rest of the current simulation step.

3.3.4 Pseudorange and Carrier

The generic pseudorange model is presented in Equation (3.35)

$$\rho = r + c[\delta t_r - \delta t_s] + I_\rho + T_\rho + \epsilon \tag{3.35}$$

where r is the true user-satellite range, δt_r and δt_s are the user and satellite clock biases respectively, I_ρ is the ionospheric error, T_ρ is the tropospheric error, and ϵ represents unmodeled error effects [39]. As previously stated, the receiver clock bias is unique to the specific receiver and estimated during the least squares position solution calculation and is therefore not modeled in this thesis. However, noise is added to some of the generated measurements discussed in the

following sections. The addition of a clock model would effect the overall noise characteristics; so in some effect it is included, but is not specific to the clock. If desired, a receiver clock model may be added to the GPS software module.

Once the true range to the satellite is found using Equation (3.1), the pseudorange can be found by adding all of the error sources previously described in Section 2.1.2 to the true range, shown in Equation (3.36).

$$\rho_k = r_k + cdt_{s_k} + I_k + T_k + \epsilon_\rho \tag{3.36}$$

The carrier phase measurement generated by the GPS software module is based on the carrier-based pseudorange shown in Equation (3.37) [39].

$$\Phi_k = r_k + cdt_{s_k} - I_k + T_k + \epsilon_\Phi \tag{3.37}$$

Note the ionosphere error is subtracted in Equation (3.37) as opposed to Equation (3.36) because the ionosphere advances instead of delaying the carrier.

3.3.5 Doppler

Doppler measurements are calculated via matrix multiplication of the differences between satellite and user positions and velocities [14] shown in Equation (3.38)

$$v_d = \begin{bmatrix} X_{s_k} - X_u & Y_{s_k} - Y_u & Z_{s_k} - Z_u \end{bmatrix} \begin{bmatrix} V_{xs_k} - V_{xu} \\ V_{ys_k} - V_{yu} \\ V_{zs_k} - V_{zu} \end{bmatrix} \tag{3.38}$$

where V denotes velocity and the x, y, and z subscripts of velocity denote the ECEF axis. The Doppler velocity v_d is then converted to Doppler frequency shift according to Equation (3.39)

$$f_d = -f_{L1}\frac{v_d}{c} \tag{3.39}$$

where f_{L1} is the L1 carrier frequency. As shown, the Doppler shifts are calculated using satellite and user positions and velocities, both of which are perfectly known to the module; therefore,

there will be no error in the calculated Doppler shift. A GPS receiver has noisy Doppler measurements; therefore noise is added to the calculated Doppler shifts to ensure measurements are representative of a receiver. The magnitude of the added noise is hand tuned such that the error characteristics of the generated measurements matches that of the Novatel receiver.

3.3.6 PVT Solution

The final position and time solution is calculated iteratively using a recursive least squares approach shown in [39] and [4]. The setup for the least squares estimation is represented by Equation (3.40)

$$\begin{bmatrix} \delta\mathbf{x} \\ \delta b \end{bmatrix} = \mathbf{G}^{-1}\delta\rho \tag{3.40}$$

where $\delta\mathbf{x}$ represents corrections to the initial position guess $\mathbf{x_0}$ and δb is the correction to the initial guess of the difference between the receiver and satellite clock bias b_0. The difference between the measured pseudorange and the range calculated using the position guess is represented by $\delta\rho$, while $\mathbf{G}$ is referred to as the geometry matrix, represented by Equation (3.41)

$$\mathbf{G} = \begin{bmatrix} -\frac{x_{s_1}-x_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{y_{s_1}-y_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{z_{z_1}-z_0}{\|\mathbf{x}-\mathbf{x_0}\|} & 1 \\ -\frac{x_{s_2}-x_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{y_{s_2}-y_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{z_{z_2}-z_0}{\|\mathbf{x}-\mathbf{x_0}\|} & 1 \\ & \vdots & & \\ -\frac{x_{s_k}-x_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{y_{s_k}-y_0}{\|\mathbf{x}-\mathbf{x_0}\|} & -\frac{z_{z_k}-z_0}{\|\mathbf{x}-\mathbf{x_0}\|} & 1 \end{bmatrix} \tag{3.41}$$

with each row representing one satellite and subscript s denoting satellite. The first three columns of each row represent the unit vector from the guessed position to the respective satellite. The least squares solution for the initial estimates is now found using Equation (3.42).

$$\begin{bmatrix} \delta\hat{\mathbf{x}} \\ \delta\hat{b} \end{bmatrix} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\delta\rho \tag{3.42}$$

New estimates of position and clock bias are then found using Equation (3.43)

$$\begin{aligned} \hat{\mathbf{x}} &= \mathbf{x} + \delta\hat{\mathbf{x}} \\ \hat{b} &= b_0 + \delta\hat{b} \end{aligned} \tag{3.43}$$

which are then used as the initial estimates and the process is repeated until the estimates converge, typically in two to four iterations [39].

Once the user position is solved for, the $\mathbf{G}$ matrix and unit vectors from the user to satellites $(\mathbf{1}^k)$ can be used to solve for user velocity. In the case of velocity, only a single least squares estimation is needed, shown in Equation (3.44)

$$\begin{bmatrix} \mathbf{v} \\ \dot{b} \end{bmatrix} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\left(\mathbf{D}\cdot\lambda_{L1} - \dot{\rho}\right) \tag{3.44}$$

where $\mathbf{D}$ is the vector of Doppler shifts for the satellites used, λ_{L1} is the L1 wavelength, and $\dot{\rho}$ is the vector of pseudorange rates found using the satellite velocities and unit vectors.

Because the velocity estimation is based on the user and satellite positions and Doppler shifts, there will be no errors in user velocity unless errors are artificially added to the pseudoranges, satellite positions, or Doppler shifts. This is due to the fact that pseudoranges and Doppler shifts are calculated based on perfectly known user and satellite positions and velocities. Because there is noise in the measurements of a real receiver, noise is artificially added to the generated measurements. Again, the magnitude of the added noise is hand tuned such that the error characteristics match that of the Novatel receiver.

3.4 IMU Software Module

3.4.1 IMU Models

Simulations of the inertial sensors used in the navigation unit are developed from the simple IMU models shown below in Equations (3.45) and (3.46).

$$g_r = r + c_r + b_r + w_{gyro} \tag{3.45}$$

$$a_y = \ddot{y} + gsin\phi + c_{\ddot{y}} + b_{\ddot{y}} + w_{accel} \tag{3.46}$$

These models, presented in [58, 16], represent the measured output of the yaw gyro g_r and the lateral accelerometer a_y. The measured output of the sensors are a combination of the true outputs r and $\ddot{y}$, plus a turn on bias c, a moving or walking bias b, and wide band sensor noise w. Note that $\ddot{y}$ includes centripetal acceleration. The lateral acceleration equation includes a term to account for the effect of gravity g when the vehicle experiences roll ϕ. The sensors are assumed to be static on a level surface when being characterized. The wide band sensor noise is assumed to be normally distributed with zero mean and sampled covariance

$$E[w^2] = \sigma^2 f_s \tag{3.47}$$

The moving bias term, or sensor drift, is modeled as a first order Gauss Markov process, in this case the output of a low pass filter with zero mean white noise input, and is outlined in (3.48)

$$\dot{b} = -\frac{b}{\tau} + w_b \tag{3.48}$$

or in discrete time

$$\begin{aligned} b_k &= \Phi_k b_{k-1} + w_{bk} \\ \Phi_k &= e^{-\frac{\Delta t_k}{\tau}} \end{aligned} \tag{3.49}$$

where τ is the time constant of the process, b is the bias, and w_b is the zero mean Gaussian random variable. Φ is the state transition maintenance, k represents the epoch, and Δt_k is

the sample interval. The first order Gauss Markov process is used because it has been extensively used in navigation and estimation to model stochastic drift characteristics present on many types of navigation system outputs [21, 45]. As seen in (3.49), the process output is a combination of the Gaussian driving noise and past values of itself. As a result, the process can be described by a Gaussian distribution and characterized with a mean and variance [58]. The Markov process accounts for the sampling frequency and has statistics shown in Equation (3.50).

$$\begin{aligned} E[b] &= 0 \\ E[b^2] &= \sigma_{bias}^2 \end{aligned} \tag{3.50}$$

where w_{bias} is given by Equation (3.51)

$$w_{bias} = \sqrt{\frac{2f_s\sigma_{bias}^2}{\tau}}\nu \tag{3.51}$$

The noise ν that drives the bias is normally distributed with zero mean and sampled covariance of one, shown in Equation (3.4.1)

$$\nu \sim N[0, 1] \tag{3.52}$$

This results in a variance for the bias, given by Equation (3.53).

$$Q_{bias} = E[w_{bias}^2] = \frac{2f_s\sigma_{bias}^2}{\tau} \tag{3.53}$$

In contrast to wide-band noise, the Markov process exhibits a non-zero time correlation because of dependence on past values. This correlation causes the process to appear as a slowly drifting bias, which is representative of the sensors being modeled [58]. An example of the Gauss-Markov process is shown in Figure 3.5.

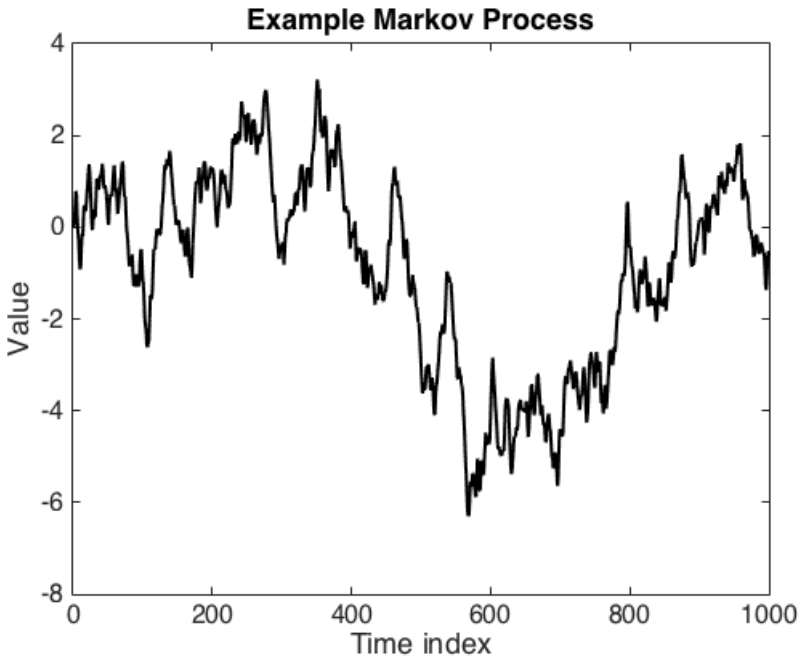

Figure 3.5: Sample Plot of a Gauss-Markov Process.

After some settling time, the Markov process autocorrelation function takes the form of Equation (3.54)

$$R_{bb}(T) = {\sigma_b}^2 e^{-T/\tau} \tag{3.54}$$

where σ_b is the variance of the Gaussian random variable in the Markov process. This autocorrelation function lends itself well to experimental identification as extraction of the magnitude and time constant of the process is straightforward given a plot of the autocorrelation [58]. The Gauss-Markov process is also referred to as exponentially correlated noise [22].

The standard deviation of the wide band sensor noise, standard deviation of the bias noise, and time constant for the Markov process used for error simulation in the IMU software module are based on the characteristics of the KVH IMU used in validation of the system. In the author's experience, the simple models do not fully capture all of the error dynamics of the KVH using experimentally determined error characteristics; however, employing an iterative approach to find error characteristics for the simple models allows for a good approximation of the KVH error dynamics. Identification techniques and characterization of the KVH and the iterative approach used to find characteristics for the simple IMU models are discussed in section 4.2.2.

3.4.2 IMU Implementation

For real time implementation, the models described in the previous section are translated into C++ code and implemented on the Linux PC and are referred to as the IMU software module. The IMU software module receives the true acceleration and rotation rate measurements from the ANVEL simulation for each epoch, and subsequently corrupts the true measurements according to Equations (3.45) and (3.46). Random noise and bias are added to each axis of the acceleration and rotation rate measurements, with the bias being calculated from (3.49). The random noise added to the measurements and the noise that drives the bias are generated using random number generators standard to C++. The accelerations from ANVEL are in the body frame and include centripetal acceleration and gravitational effects.

Outputs from the IMU simulation are also quantized due to the fact the KVH only outputs the measurement from each axis as a 16 bit floating point number. The measurements generated by the IMU software module are stored as double precision numbers to maintain high precision, but are published as 16 bit floats so the data is quantized to the same level as the hardware KVH IMU.

The IMU software module is wrapped in the ROS environment to make use of the fact the true measurements from ANVEL are already in the ROS environment. Furthermore, the KVH IMU being simulated is already sampled by code wrapped in the ROS environment. Implementation in the ROS environment also provides methods to accurately regulate the speed at which the code runs so that measurements are output at the same rate as from the real KVH. The truth measurements are corrupted by the error models and then packaged into the same ROS topics as the hardware KVH IMU measurements and subsequently published to the ROS database. The IMU software module is designed such that a navigation algorithm which uses KVH measurements as ROS topics may use measurements from the IMU software module interchangeably.

3.5 WSS Software Module

Counting encoder pulses, along with knowledge of encoder resolution and wheel radius, allows the user to determine the vehicle's velocity using (3.55)

$$v_x = \frac{\Delta count}{sample} \times \frac{sample}{\Delta t} \times res \times 2\pi \times R_w \tag{3.55}$$

where v_x is the vehicle's longitudinal velocity, $\Delta count$ is the number of pulses read during the current sampling interval, Δt represents the sampling interval, res is the resolution of the encoder in counts per revolution, and R_w is the radius of the wheel the encoder is mounted to. Because the encoders used by the GAVLab navigation unit output encoder counts when sampled, the WSS software module is designed to output encoder counts representative of the true wheel speed. The module takes perfect wheel speed information from ANVEL, converts to encoder counts, and corrupts with quantization error. Rearranging (3.56) yields Equation (3.56)

$$\frac{\Delta count}{sample} = v_x \times \frac{\Delta t}{sample} \times res \times 2\pi \times r \tag{3.56}$$

allowing the calculation of the required number of encoder counts to represent the true wheel speed from ANVEL for the given sampling time. The counts for the current epoch are added to the total number of counts and published via the same ROS topics as the encoders employed by the GAVLab navigation unit. Because the code that samples the hardware WSS is C++ wrapped in the ROS environment, the WSS software module is written in C++ and wrapped with ROS so that WSS software module outputs are interchangeable with measurements taken from the hardware WSS. Like the IMU softwre module, the WSS software module also uses the ROS framework to regulate speed, ensuring that measurements are output at the desired rate. The sampling frequency for the WSS used in Auburn's navigation unit varies, but is typically set to be 100 Hz. Therefore, the frequency for the WSS model was set at 100 Hz but can be modified as necessary.

Because a hardware encoder will only output an integer number of counts, the calculated encoder counts must be rounded down to the nearest integer, thus quantizing the data. The

partial count leftover from rounding is saved and added to the required number of counts for the next epoch so that the real distance represented by the partial count is not lost. ANVEL uses realistic tire models and friction coefficients, therefore errors in wheel speed due to slip are already accounted for in the true wheel speed received by the WSS module.

3.6 GPS Hardware Module

The GPS hardware module is made possible through the use of a Spectracom GSG-6 Series Global Navigation Satellite System (GNSS) simulator. The Spectracom simulator makes use of the Real-Time Signal Generation (RSG) software add-on from Spectracom to generate RF signals in real-time. To facilitate the GPS hardware module, a GPS hardware node is written in C++ and wrapped in the ROS framework. This allows the node to readily subscribe to the GPS data already being published by the GPS software module. The GPS hardware node subscribes to the true user position and velocity published by the GPS software node and uses Spectracom RSG commands to output the simulated position and velocity. The Spectracom takes the commanded position and velocity and generates representative RF signals. The RF signals generated by the Spectracom are fed directly into the Novatel hardware receiver used by the navigation unit, which computes a solution that is published to the ROS database via the Novatel's driver. The flow of the GPS hardware module can be seen in Figure 3.6

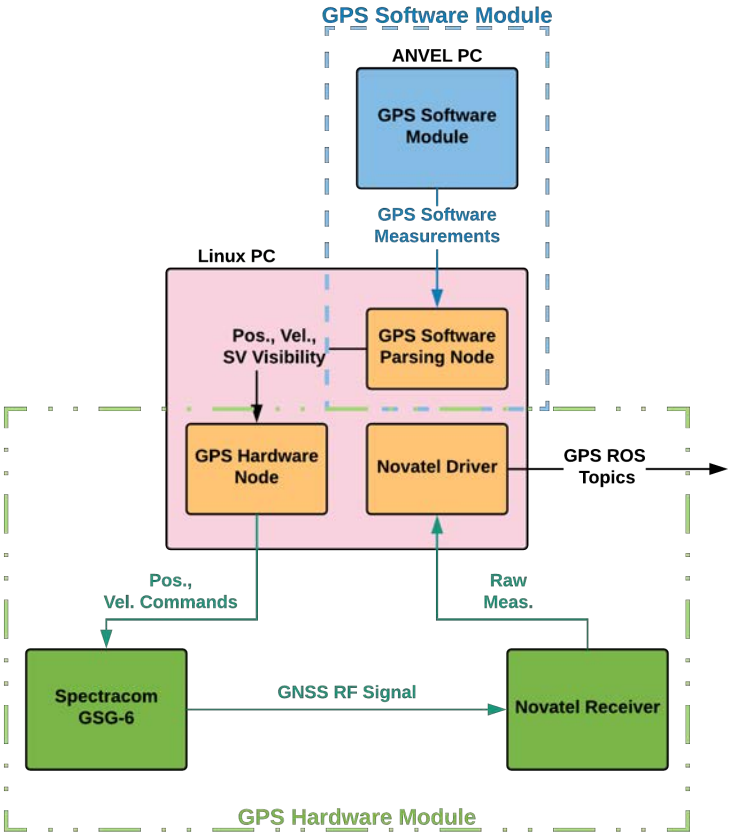

Figure 3.6: GPS Hardware Module Flow.

Because the Spectracom's signals are used by a Novatel receiver, which is then sampled by the same driver as a Novatel receiver in live sky conditions, the GPS hardware module outputs are inherently interchangeable with live sky Novatel outputs. To make the simulation as real as possible, broadcast ephemeris and environmental data from [11, 12] is uploaded to the Spectracom for the desired run period.

The Spectracom has an inherent delay in producing the RF signal representative of the desired position and velocity of at least 130 ms. To minimize the delay, Spectracom recommendeds syncing commands with the internal cycle of the simulator at 10 Hz [52]. In order to sync with the Spectracom, a query is sent to the Spectracom just after commanding position and velocity. This query only allows new position and velocity commands to be applied at the end of the Spectracom internal cycle. In practice, the author found the delay to be closer to 230 ms. An example of the delay can be seen in Figure 3.7.

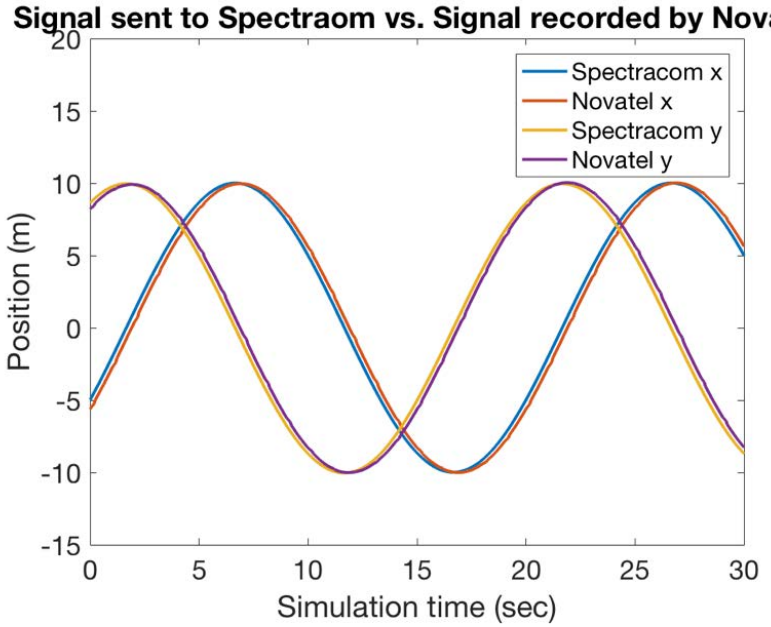

Figure 3.7: Delay in GPS Hardware Module.

To generate the above Figure, a circular path is simulated via a talker node. The talker node outputs position and velocity measurements to be used by the GPS hardware node. The GPS hardware node receives velocity and position from the talker node and commands those to the Spectracom which is then sampled by the Novatel as described. The time-stamped positions from the circular path are compared to the time-stamped positions output by the Novatel. Both positions are time-stamped using the Linux PC internal clock and thus are readily compared. The average time offset between position generation in the talker node and that position being represented by the Novatel in the run shown is 230 ms. Additionally, the Spectracom simulation may only begin at the beginning of each minute, increasing the difficulty of replicating live data.

3.6.1 Ray Tracing

The Spectracom simulator has the ability to perform ray tracing when provided a map of the environment; however, the environment model a user can load on the Spectracom is extremely limited in size. Because of this, the GPS hardware module employs the ray tracing results from ANVEL in the GPS software module shown in Figure 3.4. Satellite availability information is published from the ANVEL simulation to the GPS hardware node on the Linux PC which is then used to command satellite power in the Spectracom. If the ray tracing determines

a satellite is blocked due to an obstacle, that satellite's power is commanded to the minimum power allowed by the Spectracom causing the GPS receiver in use to lose track of the satellite. The Spectracom does not allow the user to command a satellite be turned completely off in the real-time operating mode. Once the blocked satellite returns to view, that satellite's power is commanded to return to its power level before being blocked, causing the Spectracom simulator to output a power level to allow the receiver to re-acquire the signal. Note this setup will also produce realistic signal re-acquisition from the GPS receiver.

3.7 IMU Hardware Module

The IMU hardware module uses the IMU software module as a base for measurement generation. However, instead of publishing IMU data as a ROS topic like the software module, the IMU hardware module outputs a serial signal representative of the KVH IMU. The generated IMU measurements are placed in packets the same way as the true KVH output measurements and then sent over serial via standard C++ serial methods. The KVH IMU being simulated employs RS-422 serial instead of RS-232, which poses an issue due to the fact that most computers, including the Linux PC used, do not have RS-422 ports. For this reason, a USB to RS-422 adapter is used to convert the packets into the proper serial signals. Quantization error for the IMU hardware module is handled in the same way as the software module by casting the corrupted measurements to 16 bits before packaging. The flow of the IMU hardware module can be seen in Figure 3.8.

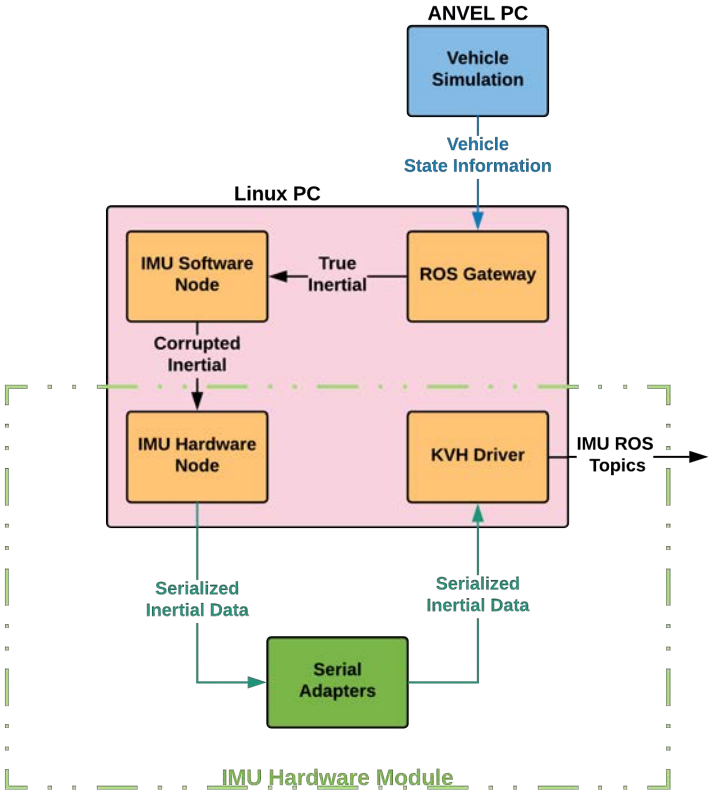

Figure 3.8: IMU Hardware Module Flow.

As previously stated, the goal of the IMU hardware module is to output a serial data message representative of the hardware IMU being emulated. To ensure this is the case, the serial output of the module is sampled by the same C++ code used to sample the actual KVH IMU. Since the IMU hardware module signal can be sampled using the same code as the KVH, a navigation unit which uses the same KVH IMU can use the output from the module interchangeably.

3.8 WSS Hardware Module

Instead of outputting encoder counts, the WSS hardware module generates quadrature encoder pulses for input into the navigation unit. Quadrature encoders output two signals, 90 degrees out of phase, both with an inverse. The two signals being out of phase allows the sampling device to determine the direction of travel. If the A signal leads B, the vehicle is

moving forward. Alternatively, if B leads A, the vehicle is moving backwards. A visualization of the signals can be seen in Figure 3.9.

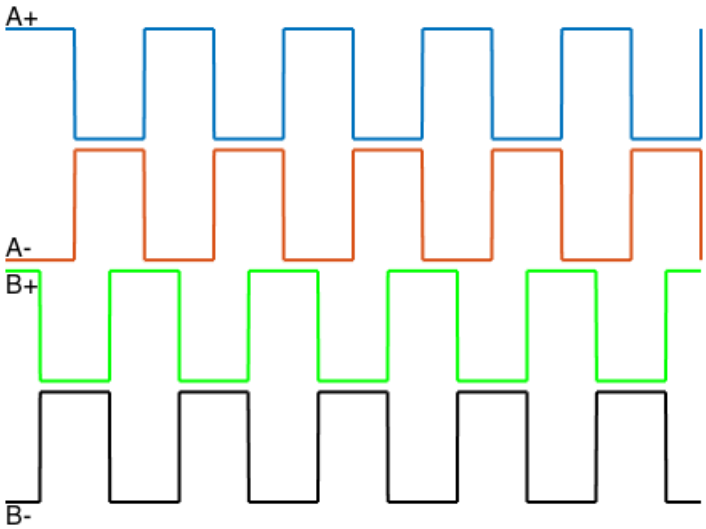

Figure 3.9: Quadrature Encoder Waveforms.

To generate waveforms shown in Figure 3.9, software for the WSS hardware module is implemented on an Arduino Due. The Due was chosen for the WSS hardware module because it has multiple Pulse Width Modulators (PWMs) and stepper motor functionality. The PWMs can be set to almost any frequency desired and with a 50% duty cycle yield a square wave. The stepper motor functionality automatically creates a phase lag of 90 degrees between the signals of specific PWMs on the board. The PWMs chosen automatically generate the inverse signal which is output on a separate pin. The combination of the original PWM, 90 degree shifted PWM, and their respective inverses combine to form the desired quadrature signal. Two sets of quadrature signals, one set for each side of the vehicle (i.e. left and right wheels used on some navigation systems), are output by the Arduino and fed into the QSB units to be sampled. The flow of the WSS hardware module may be viewed in Figure 3.10 below.

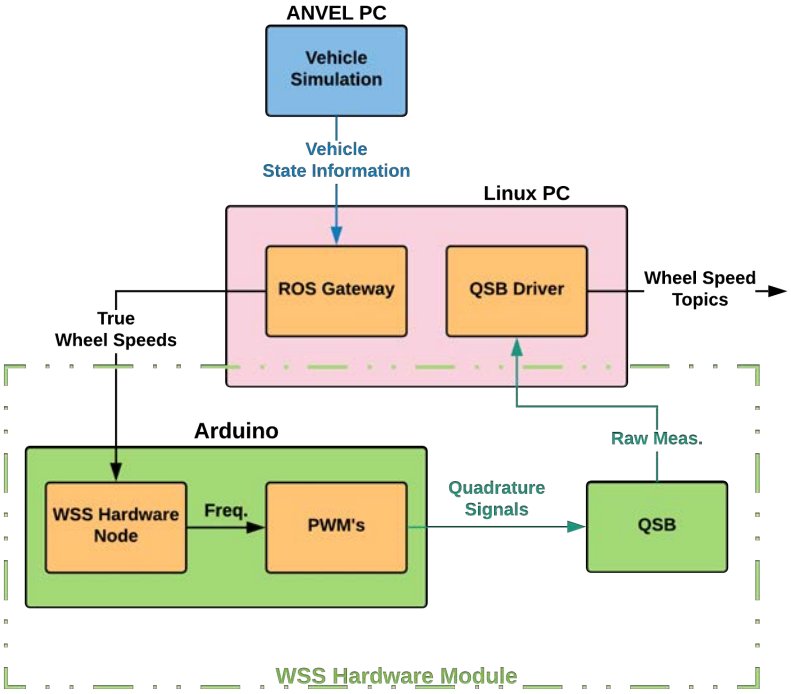

Figure 3.10: Wheel Speed Sensor Hardware Module Flow.

Communication between the Arduino and the rest of the system is handled by the ROS serial software package. The ROS serial package is available from ROS and allows the code on the Arduino to subscribe directly to the ROS topics published by the ANVEL ROS gateway. After the wheel speed topic is received by the Arduino, the on-board software calculates the required number of encoder counts to represent the wheel speed for the given epoch. If the wheel speed is not constant between epochs, the code will change the PWM frequency and a partial pulse would be lost. Therefore, the code rounds down the required number of pulses as the software module does with counts, taking care of quantization error. The partial pulses that are rounded off at each calculation are added to the next sample period's desired number of pulses. Keeping track of the partial pulses is important because these partial pulses represent some real distance the vehicle has traveled. Repeatedly throwing away the partial pulses will result in a non-zero mean velocity error and underestimation of distance traveled.

Next, the WSS hardware module calculates the frequency required for a square wave to generate the desired number of pulses over the given sample time

$$\frac{\Delta count}{sample} = v_x \times \frac{\Delta t}{sample} \times res \times 2\pi \times R_w \tag{3.57}$$

and the PWM's are set to the determined frequency. The pins of the PWM's are wired to the same QSB adapters used to sample the hardware encoders, which are then sampled using the same code as with the hardware encoders. Since the WSS hardware module is sampled using the same code as the hardware encoders, it can be readily used in place of the hardware encoders. The only modification to the navigation unit required is to simply plug the wires from the Arduino PWM's into the QSB adapters.

3.9 Conclusions

This chapter introduced the overall system architecture and how the developed system interacts with ANVEL using plugins. The plugins were used to gather vehicle state information such as position, velocity, and accelerations, which were then used as inputs to all of the developed software and hardware modules. Next, the GPS, IMU, and WSS software modules which output ROS topics to mimic the actual sensors were introduced and their respective models discussed. The GPS, IMU, and WSS modules were then introduced to output signals representative of the real sensors being emulated. The GPS hardware module used a Spectracom simulator to output GPS RF sampled by a receiver, while the IMU hardware module output serial data representative of a real KVH IMU, and the WSS hardware module output quadrature encoder pulses to mimic wheel encoders. The testing and validation of the developed software and hardware modules is discussed in the following chapter, as well as some implementation issues.

Chapter 4

Testing and Validation of Simulation Environment

In this chapter, performance of both the software and hardware modules is analyzed by comparison to experimental data collected with the sensors used by the GAVLab navigation unit. Furthermore, the simulated data is compared to truth sensors and truth output data from ANVEL in order to quantify performance. First, experimental test setups are introduced, followed by the comparison of outputs from the software modules and finally the hardware modules. All of the modules are examined in both static and dynamic test scenarios. It is important to note that validation of ANVEL is outside of the scope of this work. ANVEL is a product developed under contract for the US Army, so it is assumed ANVEL was validated before being delivered. This thesis simply takes ANVEL outputs to produce the emulated sensor data.

4.1 Test Setups

In order to validate each module described in the previous chapter, both static and dynamic test modes are employed. The static test routes are used for initial validation of the GPS and IMU hardware and software modules. Static testing allows for initial proof of concept and model validation as well as perfect knowledge of the true states such as acceleration, velocity, and position. Static testing of the IMU modules is performed by placing an IMU on a level surface and comparing the statistics of the data output by the IMU modules to a static data set from the KVH IMU. The setup for static testing of the GPS modules is detailed in Section 4.1.2.

4.1.1 Experimental Data Collection Setup

All dynamic experimental data collection is performed using the GAVLab's 2003 Infiniti G35. The G35 is outfitted with Septentrio PolarX2e@ 3-axis GPS receiver, Honeywell eTalin, wheel speed sensors, GAVLab navigation unit, and Linux Advantech PC for data collection. The Linux PC used in experimental data collection is the same Linux PC employed in the developed HIL/SIL environment. The test vehicle is shown in Figure 4.1.

Figure 4.1: The GAVLab's 2003 Infiniti G35.

As seen in the above figure, there are 3 GNSS antennas arranged at a right angle mounted to the top of the G35. These three antennas provide roll, pitch, and yaw measurements and are fed to the Septentrio receiver located in the trunk which provides Real-Time Kinematic (RTK) GPS positioning using RTK corrections from the Auburn Continuously Operating Reference Station (CORS) or the GAVLab's RTK base station. The antenna on the driver's side above the rear axle is the main antenna used for the Septentrio and is also fed into the GAVLab navigation unit using a powered GPS splitter. Figure 4.2 highlights the wheel speed encoders mounted to the rear wheels of the G35.

Figure 4.2: Wheel Encoders on the G35.

4.1.2 Static GPS

Static GPS data collection was performed using a static antenna on the roof of the Woltosz Engineering Research Laboratory building on Auburn University's main campus. An overhead view of the Woltosz is presented in Figure 4.3, with the antenna location denoted by the red dot on the roof of the building. This antenna is used because the location is precisely known from previous testing using GPS data with RTK corrections. Again, RTK corrections are provided in real time by the CORS network. The antenna is not moving so the position should be constant from one time epoch to another. The velocity should be zero at all times, excluding errors; therefore, the static GPS test setup allows for easy initial model validation of position and velocity. As can be seen in the Figure, another advantage of this antenna location is the antenna's height above the ground; the antenna has a clear view of the sky with no obstructions to block satellites or reflect their signals causing multipath errors.

Figure 4.3: Overhead View of Antenna of Roof of Woltosz.

The indoor portion of the static GPS test setup can be seen in Figure 4.4. The signal from the antenna on the roof is passed through a powered signal splitter and fed to a Novatel FlexPak and a Novatel DL-V3, all pictured on the left side of the Figure. Both receivers are sampled using the Linux PC shown on the right using C++ code written by the GAVLab. The DL-V3 is provided RTK corrections from the Auburn CORS station and used as a truth signal. The FlexPak receiver contains a Novatel 628 board like the one in the navigation unit, and its measurements are used to compare to the outputs of the GPS modules. Static testing with the Spectracom only requires the FlexPak be unplugged from the roof antenna and connected to the cable from the Spectracom. The FlexPak is then sampled in the same manner as if it were connected to the roof antenna.

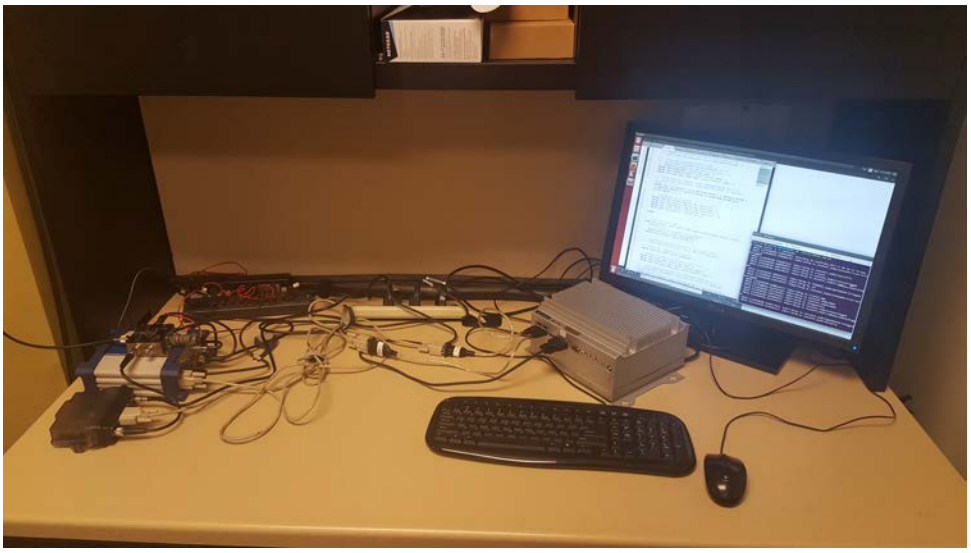

Figure 4.4: GPS Data Collection Setup in Woltosz.

4.1.3 HIL/SIL Test Setup

The entirety of the HIL/SIL system developed in this work fits on a single desk and can be seen in Figure 4.5. The Spectracom simulator is on the far left and pictured next to a Novatel receiver. Both connect to the Linux PC seen on the right side of the desk which houses all sensor drivers and developed modules. The Arduino used for WSS pulse generation is seen next to the mouse in the center and is connected to the Linux PC using a USB cable. The PWM output pins are wired to a QSB adapter plugged into a USB port on the Linux PC. Finally, the ANVEL PC is represented by the screen on the left. The ANVEL PC runs the ANVEL simulation and is connected to the Linux PC via ethernet connection.

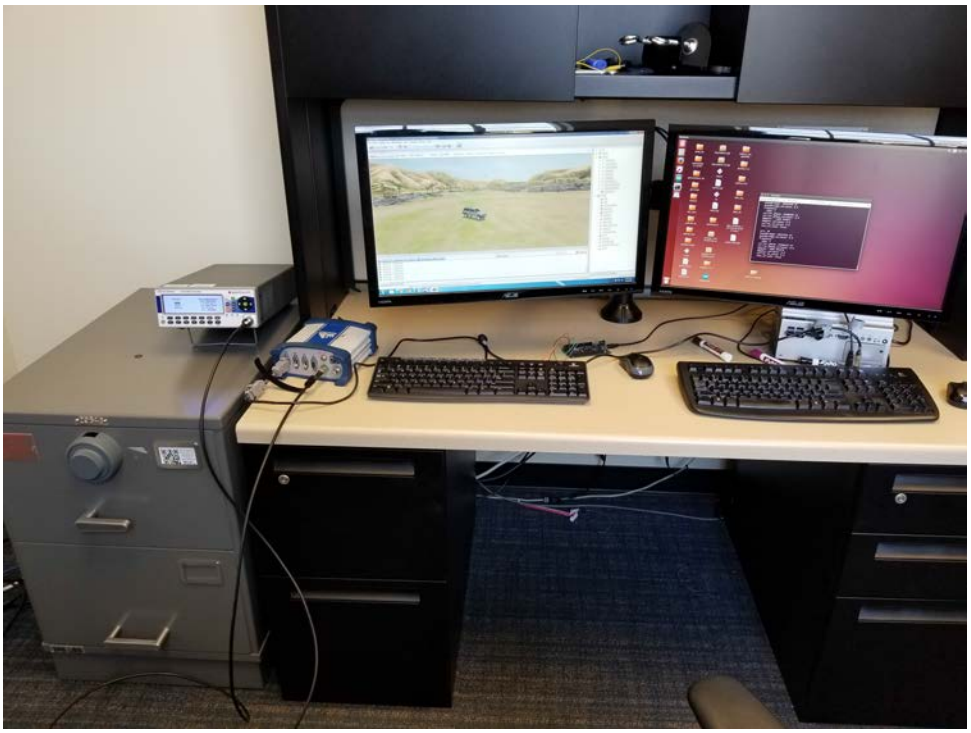

Figure 4.5: HIL/SIL Test Setup.

4.1.4 NCAT Test Route

Auburn University's National Center for Asphalt Technology (NCAT) test track is used as a dynamic data collection environment and can be seen in Figure 4.6. The NCAT track is a closed track of 1.7 miles with known bank angles in the turns. The track serves as a benign test environment on which there is no traffic or traffic laws; therefore, any maneuvers may be tested. The track is outlined in red with its start/stop point marked. Additionally, RTK corrections are provided to the GPS sensors on the test vehicle when testing at the NCAT track using the GAVLab base station depicted by the blue dot in the Figure. RTK corrections are calculated by a Novatel ProPak v3 and sent over radio link to the test vehicle.

Figure 4.6: Overhead View of NCAT Track.

4.1.5 Ray Tracing Test Setup

The ray tracing test setup is used to test the ray tracing capabilities of the GPS modules. The route consists of the two buildings highlighted in Figure 4.7, which are located at the NCAT test track. These two buildings are chosen as they are relatively close to one another and face in opposite directions, allowing the test vehicle with GPS antenna/receiver to drive from one to the other in a short amount of time.


Figure 4.7: Overhead View of Ray Tracing Test Route.

When using the ray tracing test route, the GPS sensor is moved from the blue dot to the red dot and vice versa. When the vehicle is close to one building, a large portion of the sky is blocked. When the vehicle is close to the other building, a large portion of the opposite side of the sky

is blocked. Blocking opposite portions of the sky in a short time frame as described causes several different satellites to go in and out of view, allowing for verification of the ray tracing functionality of the simulation.

4.1.6 Auburn Test Route

The second dynamic test route employed is through downtown Auburn and used to test the performance of the entire system in a real-world driving scenario. The downtown Auburn route provides stop and go traffic, as well as buildings and trees that block out portions of the sky and reflect GPS signals, providing an opportunity to test the ray tracing functionality of the GPS modules. An overhead view of the downtown Auburn test route, outlined in red with start and stop location denoted, can be seen in Figure 4.8, while a street level view demonstrating some of the obstructions present in the test route can be seen in Figure 4.9.

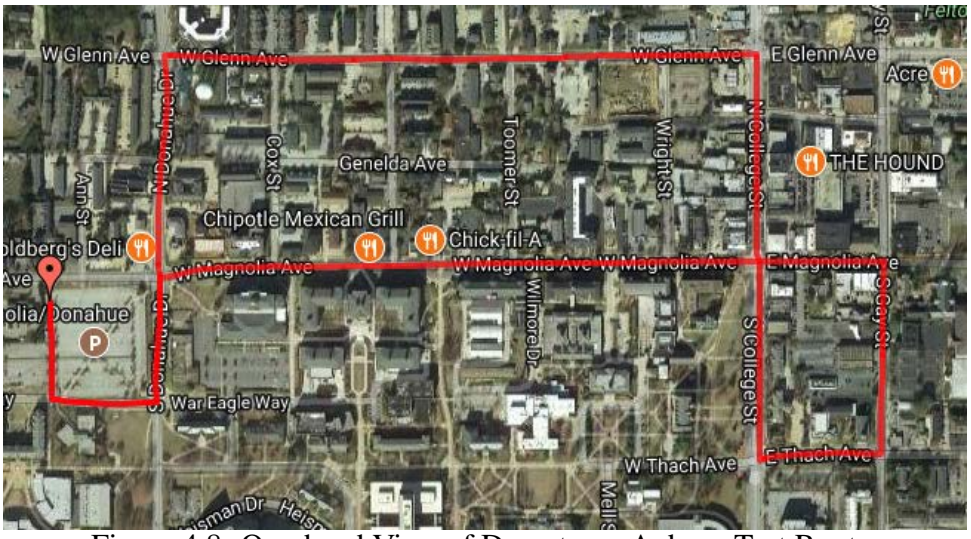

Figure 4.8: Overhead View of Downtown Auburn Test Route.

Figure 4.9: Street Level View of Portion of Downtown Auburn Test Route.

4.1.7 ANVEL Test Routes

The large parking lot environment, seen in Figure 4.10, is a large, flat asphalt terrain included with ANVEL and is used for dynamic testing of all modules. As seen in the Figure, there are no obstructions or variations in terrain to introduce errors to the modules and thus provides a benign test environment. Because the environment is so large, it is well suited for testing the WSS modules allowing the vehicle to reach high speeds and maintain those speeds for extended amounts of time. The large parking lot environment is also used to test the GPS software module for static scenarios. The origin of the environment can be changed to simulate any location.

Figure 4.10: ANVEL Large Parking Lot Environment.

Both the ray tracing test route and downtown Auburn test routes depicted in the previous sections were converted into 3-D environments and imported into ANVEL for use in system validation. The ANVEL ray tracing test route is a modification of the large parking lot environment with its origin changed such that the environment represents the area around the NCAT track. Two large walls were added to the environment to simulate the sides of two buildings, representative of the buildings shown in Section 4.7. These walls are placed in the environment so that the ray tracing portion of the GPS module may be tested and validated against live data. A 3-D model of downtown Auburn was also created to replicate the Auburn test route discussed in Section 4.1.6. An overhead view of the downtown Auburn environment in ANVEL can be seen in Figure 4.11. The environment was created using Blender and the Open Street Maps (OSM) plugin, allowing the easy importing of buildings with accurate locations into the environment. However, most of the imported buildings do not have accurate heights so adjustments were made to match the real buildings. A second issue with creating an environment in Blender is the roads are not well defined, so setting the friction coefficients accurately is difficult. As a result, the Auburn test route is not used for testing of the WSS modules. Once the environment is made in Blender, it is then imported into ANVEL following steps outlined on ANVEL's website [44].

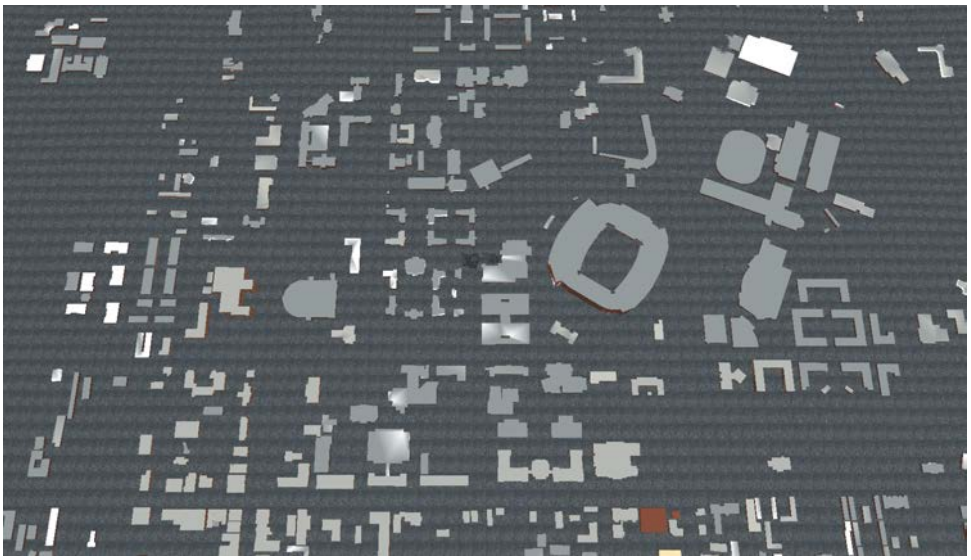

Figure 4.11: OSM Auburn Model in ANVEL.

4.2 Hardware Sensor Characterization

In order to obtain error characteristics for development and validation of each of the developed modules, the GPS receiver, IMU, and WSS used in the GAVLAb navigation unit must be characterized. The following subsections introduce sensor characterization techniques for GPS, IMU, and WSS and present the results from these techniques. The error dynamics resolved from sensor characterization are used as guides during module development in Chapter 3 and as benchmarks for comparison beginning in Section 4.3.

4.2.1 GPS Characterization

GPS measurement errors are mostly due to effects external to the receiver as described in Section 2.1.2. These errors are common to all GPS receivers; however, more expensive receivers are able to mitigate some of the errors through advanced signal processing techniques. These techniques are avoided in this work and instead data is gathered with the receiver from the navigation unit in various static and dynamic scenarios for error characterization. Truth in the static scenarios for position and velocity are the known Woltosz antenna position and velocities, while truth pseudorange and Doppler data is obtained from the Novatel receiver with RTK corrections. Figures 4.12 and 4.13 present pseudorange and Doppler error plots for satellites 3 and 23.

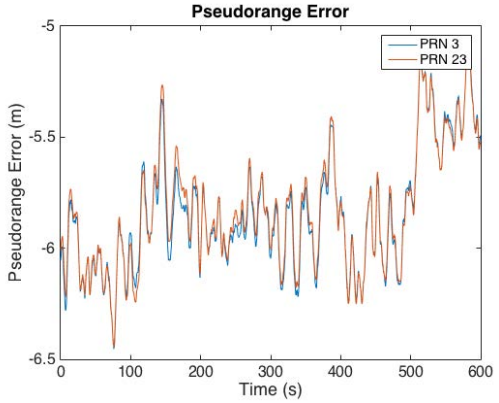

Figure 4.12: Standalone Novatel Pseudorange Errors

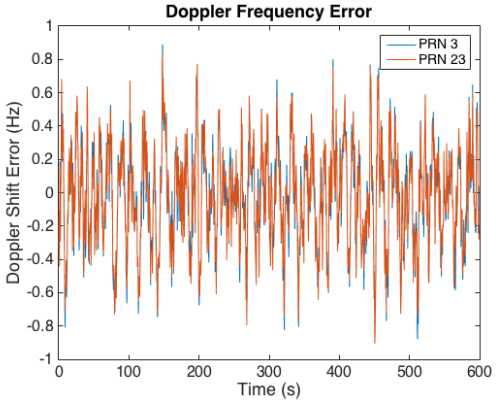

Figure 4.13: Standalone Novatel Doppler Errors

The pseudorange and Doppler errors are very similar in magnitude and follow the same trends. A Doppler error of 1Hz corresponds to a pseudorange rate error of 19 cm/s. Position and velocity error plots for the standalone Novatel receiver are shown in Figures 4.14 and 4.15.

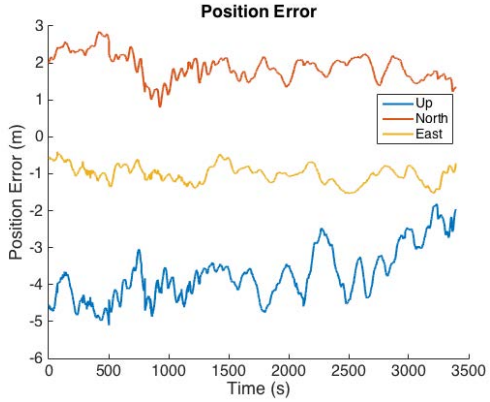

Figure 4.14: Standalone Novatel Position Errors.

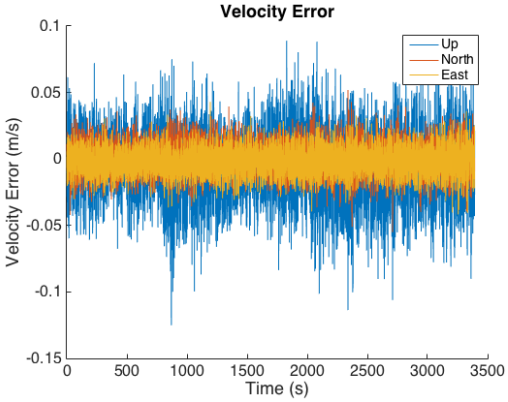

Figure 4.15: Standalone Novatel Velocity Errors.

Position and velocity errors are simply the difference in Novatel reported values and the known location and velocity of the antenna. A summation of the rms error for each of the measurements shown is presented in Table 4.1. Because the characterization data set is only one hour in length, the slowly changing GPS errors such as satellite ephemeris error with time constants of about an hour cannot be fully characterized. The values shown in Table 4.1 are used as performance guidelines when tuning the errors in the GPS software module.

Table 4.1: Standalone Novatel Receiver Measurement Errors

Measurement	RMS Error
East Velocity	1.0 cm/s
North Velocity	1.1 cm/s
Up Velocity	2.6 cm/s
East Position	1.0 m
North Position	1.9 m
Up Position	3.8 m
PRN 3 L1 Pseudorange	5.9 m
PRN 3 L1 Doppler Shift	0.29 Hz

4.2.2 KVH 1725 Characterization

During initial algorithm development and concept verification, the sensor characteristics from the respective data sheets are used to model the sensors. However, to improve the accuracy of the simulation, the sensors used in the hardware implementation are characterized and their statistics are used in the simulation. For the purpose of this work, Allan variances are conducted on static sensor data to determine the true characteristics of the KVH IMU used in the navigation unit. The Allan variance is a technique first introduced in [1] to characterize the errors of atomic clocks. However, the technique has proven to be useful in determination of IMU error characteristics. The Allan deviation plot for the Z gyroscope of the KVH 1725 can be seen in Figure 4.16 using 24 hours of static data sampled at 100 Hz.

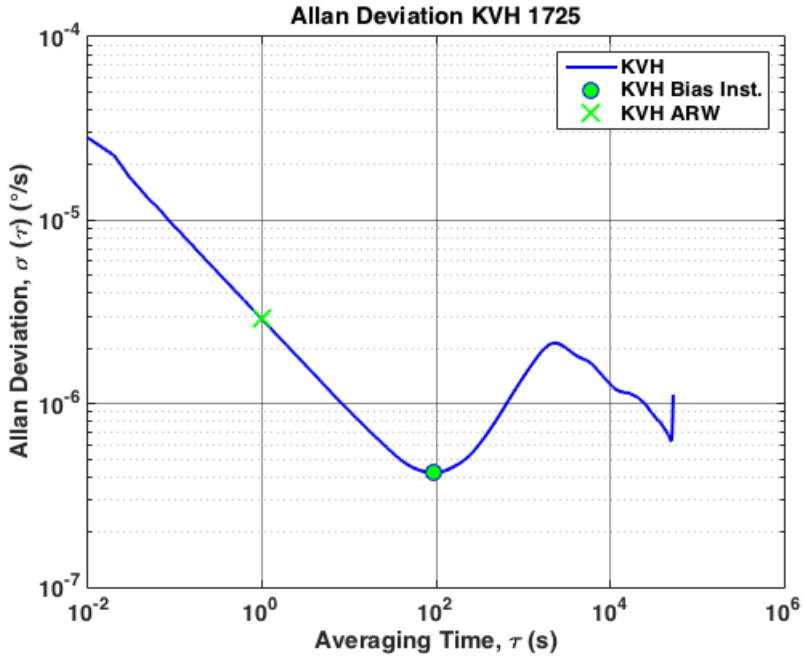

Figure 4.16: Allan Deviation of the KVH 1725 Gyroscope.

The major points of analysis on the Allan deviation plot for the purposes of this work are the section where the slope is -1/2, the flat point around the minimum, and the portion where the slope is +1/2. The section of the plot where the slope is -1/2 represents the Angular Random Walk (ARW), or wideband noise, of the gyroscope, and the flat area around the minimum represents the bias instability of the gyroscope. The portion of the plot where the slope is +1/2 represents exponentially correlated noise, or a first order Markov process [58]. To find the value for the ARW, a line is fit to the portion of the plot with a slope of -1/2. Where this line intersects τ equal to 1 is the ARW. Applying this to Figure 4.16 yields a ARW of 2.96×10^{-6} rad/s/$\sqrt{\text{Hz}}$. The manufacturer specifies the ARW as no worse than $1^{\circ}/hr/\sqrt{\text{Hz}}$. After unit conversion, the manufacturer specified ARW becomes 4.84×10^{-6} rad/s/$\sqrt{\text{Hz}}$, meaning the IMU used is performing better than specified. The bias instability read from the plot is 4.22×10^{-7} rad/s. Experimentally determined values for ARW, Velocity Random Walk (VRW), and bias instability for each accelerometer and gyroscope are compared to the manufacturer specifications in Tables 4.2 and 4.3.

Table 4.2: Experimentally Determined KVH Accelerometer Parameters

Parameter	Axis	Manufacturer Specification	Experimental Value
Velocity Random Walk ($m/s^2/\sqrt{Hz}$)	X	1.09×10^{-3}	7.38×10^{-4}
	Y	1.09×10^{-3}	8.35×10^{-4}
	Z	1.09×10^{-3}	8.04×10^{-4}
Bias Instability (m/s^2)	X	9.09×10^{-4}	3.72×10^{-4}
	Y	9.09×10^{-4}	4.54×10^{-4}
	Z	9.09×10^{-4}	3.91×10^{-4}

Table 4.3: Experimentally Determined KVH Gyroscope Parameters

Parameter	Axis	Manufacturer Specification	Experimental Value
Angular Random Walk ($rad/s/\sqrt{Hz}$)	X	4.84×10^{-6}	2.96×10^{-6}
	Y	4.84×10^{-6}	2.89×10^{-6}
	Z	4.84×10^{-6}	2.96×10^{-6}
Bias Instability (rad/s)	X	4.84×10^{-7}	4.21×10^{-7}
	Y	4.84×10^{-7}	4.22×10^{-7}
	Z	4.84×10^{-7}	3.59×10^{-7}

Due to the use of the simple sensor models described in Section 3.4.1, the bias instability value cannot be directly used in the model. Instead, an autocorrelation is performed on the gyroscope data in an attempt to resolve the time constant and standard deviation of the noise of the Markov process used to model the sensor. Because the sensor output is a combination of several error sources, the Markov process must be isolated for identification. Approximate isolation may be achieved by low-pass filtering the raw sensor output to removed the high frequency noise. The lower frequency errors may still exist in the data after filtering, increasing the difficulty of identification. Also, identifying the correct cut-off frequency requires some knowledge of the approximate time constant to be identified, making identification an iterative process [58]. The autocorrelation of the KVH Y gyroscope data is shown in Figure 4.17.

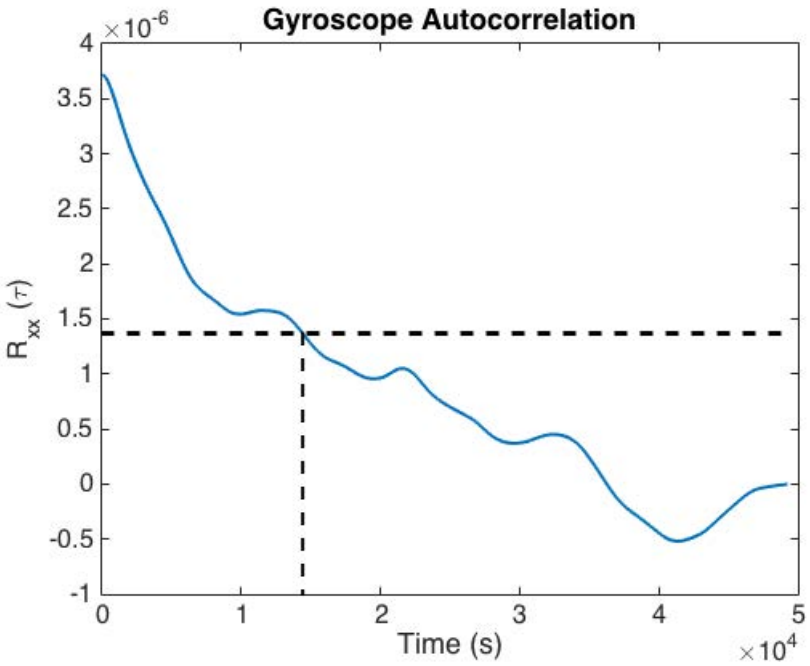

Figure 4.17: Autocorrelation of the KVH 1725 Gyroscope.

Equation 3.54 gives the autocorrelation function of the Markov process, which is an exponential decay with initial magnitude σ_b^2 and time constant τ. The initial magnitude of the Markov process is simply the y intercept, while the time constant is the time shift read from the point at which the autocorrelation decays to $\frac{1}{e}$, or 36.8% of its original magnitude [16]. Examination of the autocorrelation plot reveals σ_b^2 is 3.7×10^{-6} rad/s^2 and the time constant is 14,413 seconds. Unfortunately, the autocorrelation accuracy is limited by the length of data set gathered and the presence of other error sources. In general, the data set must be of length several times longer than the time constant. The autocorrelation shown was generated using the most stable 14 hours of data from the 24 hour dataset. Difficulties in identification of the Markov process are discussed in [58]. The Y gyroscope of the KVH was chosen because it produced the best autocorrelation plot.

4.2.3 Wheel Speed Sensor Characterization

Characterization of the wheel speed sensors is accomplished by collecting wheel speed data during many test runs around the NCAT test track. The data collected to characterize the WSS consists of wheel speeds from both rear wheels of the vehicle and RTK GPS data used as truth. Error characteristics for the WSS are obtained by differencing the wheel speeds reported

by the WSS with the RTK GPS velocity, which has an accuracy of about 2-3 cm/s. During testing it was observed that the WSS error grew quadratically with increasing speed, shown in Figure 4.18. The wheel speed error shown is calculated as the difference between the reported wheel speed and the RTK GPS speed in the straight portions of the test track. The growing errors are due to changes in the effective wheel radius and the fact the tires must produce more force at higher speeds to overcome increasing air drag, thus increasing wheel slip.

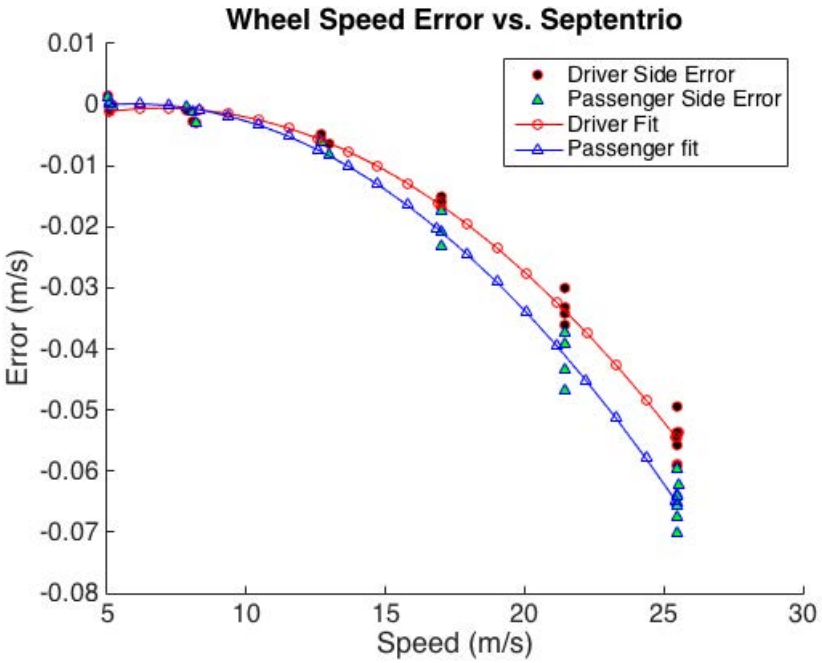

Figure 4.18: Wheel Speed Error Growth vs Speed.

The error growth with speed observed in Figure 4.18 for the driver side can be fit by Equation (4.2.3).

$$\epsilon(s) = -0.0000315s^2 + 0.0009794s + -0.0081363 \tag{4.1}$$

where s is the longitudinal speed of the vehicle. The error becomes more negative as speed increases, meaning the Septentrio speed is higher than the speed reported by the wheel speed sensors. This is the opposite trend than expected due to the fact the drive wheels must produce more force at higher speeds to overcome air resistance, thus slipping more.

Figure 4.19 represents a constant speed portion of one of the previously mentioned characterization runs around the NCAT test track. As seen in the plot, the delta count measurement, which is converted to wheel speed by Equation (3.55), oscillates between measurements creating wheel speed errors greater than one half quantization step seen in Figure 4.20. The oscillation is due to clock jitter within the QSB and the computer used to sample the sensors.

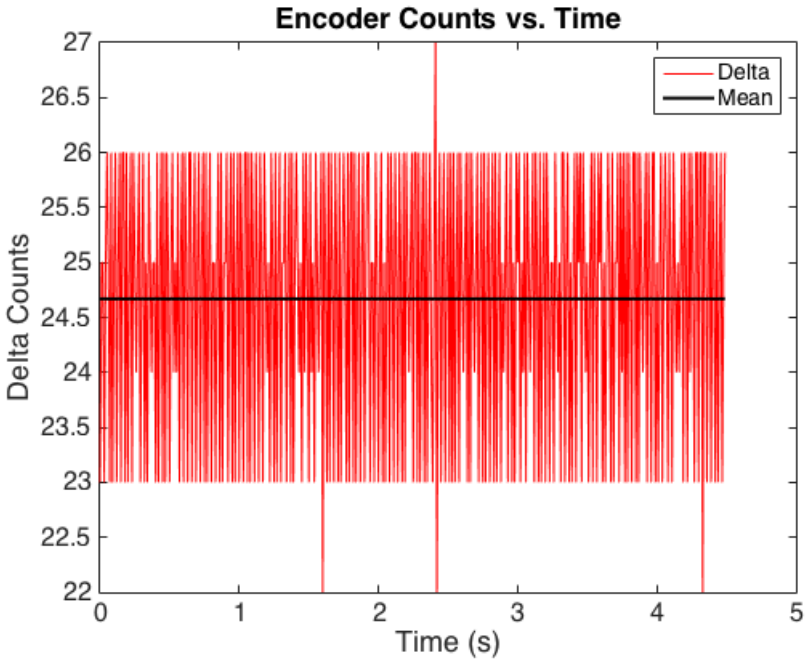

Figure 4.19: Oscillation Observed in Delta Encoder Counts.

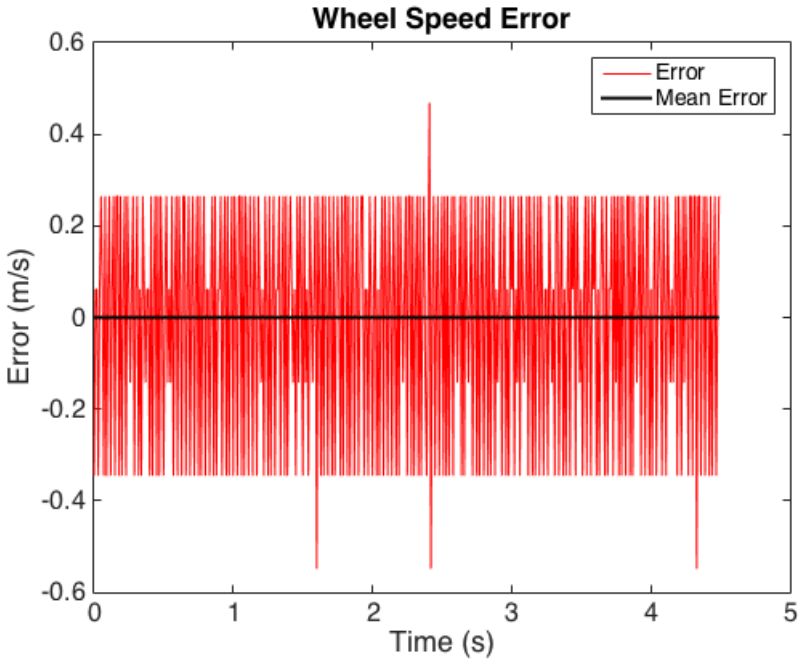

Figure 4.20: Wheel Speed Error Caused by Oscillation in Encoder Counts.

The wheel radius used for the simulated data shown is 0.323 m. With 1,000 ticks per revolution, one quantization step is 0.00203 m, with a maximum quantization error of 0.001017 m. Because the WSS operate at 100 Hz, the maximum quantization velocity error is 0.1017 m/s, clearly exceeded in Figure 4.20.

Through examination of the oscillation over several test runs at various constant speeds, it was determined the magnitude of the wheel speed oscillation is correlated to speed, shown in Figure 4.21.

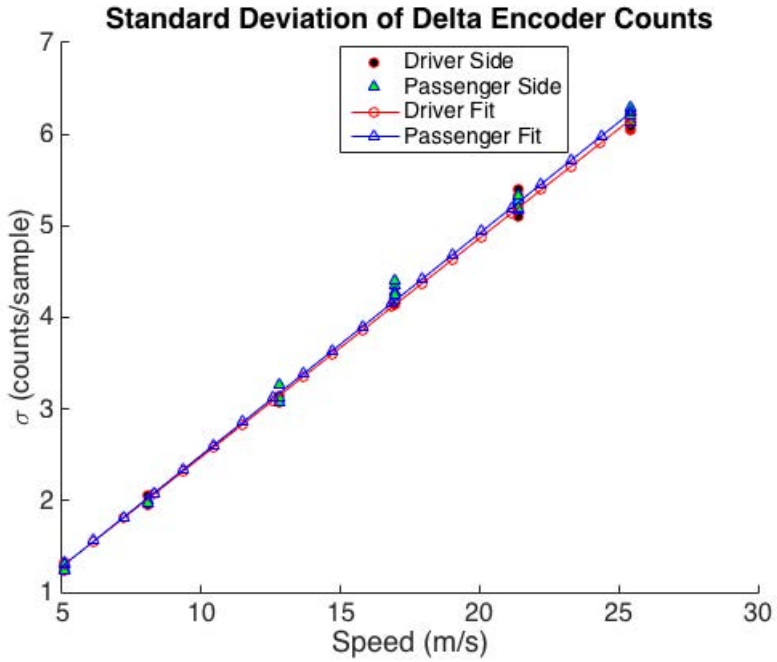

Figure 4.21: Wheel Speed Standard Deviation Growth vs Speed.

The standard deviation of encoder counts is found to increase with speed according to Equation (4.2).

$$\sigma_{WSS}(s) = 0.04860s + 0.08796 \tag{4.2}$$

Error due to wheel slip is handled by ANVEL during the physics simulation. The oscillation observed in the data is taken as a design parameter for the WSS software and hardware modules.

4.3 Software Module Validations

The error characteristics found in the previous sections are now applied to the developed models for the GPS, IMU, and WSS. In the case of the GPS and IMU modules, an iterative

approach to the application of errors is required so that the final outputs are representative of the sensors being modeled. The testing, model validation, and verification of the interfaces for each module is discussed in the following subsections.

4.3.1 GPS Software Module

To test the GPS software module, the same static data set used to characterize the Novatel in Section 4.2.1 is simulated using the GPS software module. The first measurement examined is pseudoranges for each SV in view. Figure 4.22 shows the pseudorange error for several of the satellites in view for the first minute during the test run.

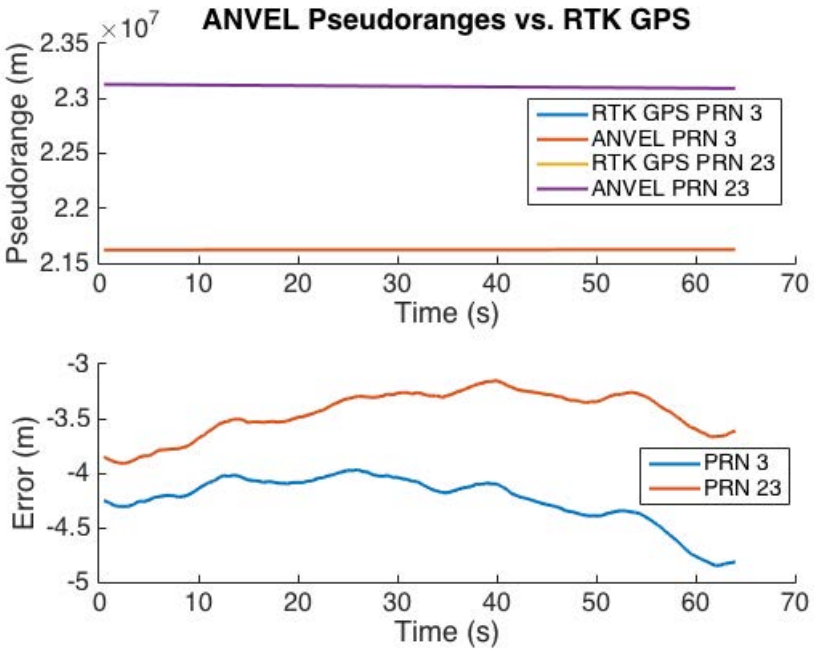

Figure 4.22: GPS Software Module Pseudoranges Compared to RTK Novatel.

The pseudorange errors for satellites 3 and 23 exhibit errors around -4.5 m and -3.5 m respectively, compared to -6 m for both when comparing standalone Novatel to RTK GPS. It is important to note that when compared to pseudoranges from a GPS receiver, the GPS software module pseudoranges will contain errors due to the broadcast ephemeris errors and the fact that the true atmospheric error calculated through the models may not exactly match the actual atmospheric error experienced by the signal traveling through the atmosphere. However, the position solution is unaffected due to the fact that the simulated pseudoranges are calculated using the true user position.

Next, the Doppler shift is examined. Figure 4.23 shows the difference between Doppler shifts generated by the GPS software module and Doppler shifts reported by the Novatel receiver with RTK. The simulated Doppler shifts are shown to be close to the measured truth Doppler shifts, within half of one Hertz of the RTK GPS for the duration of the test run, similar to the standalone Novatel. Because the software module calculates Doppler shift from perfect user velocity and the calculated satellite velocities, the errors shown are due to broadcast ephemeris errors and receiver clock drift. The position and velocity solution errors for the first minute are shown in Figures 4.24 and 4.25. The errors are denoted as limited due to the fact atmospheric error is the only error added to the measurements, resulting in smaller error magnitudes than experimental data.

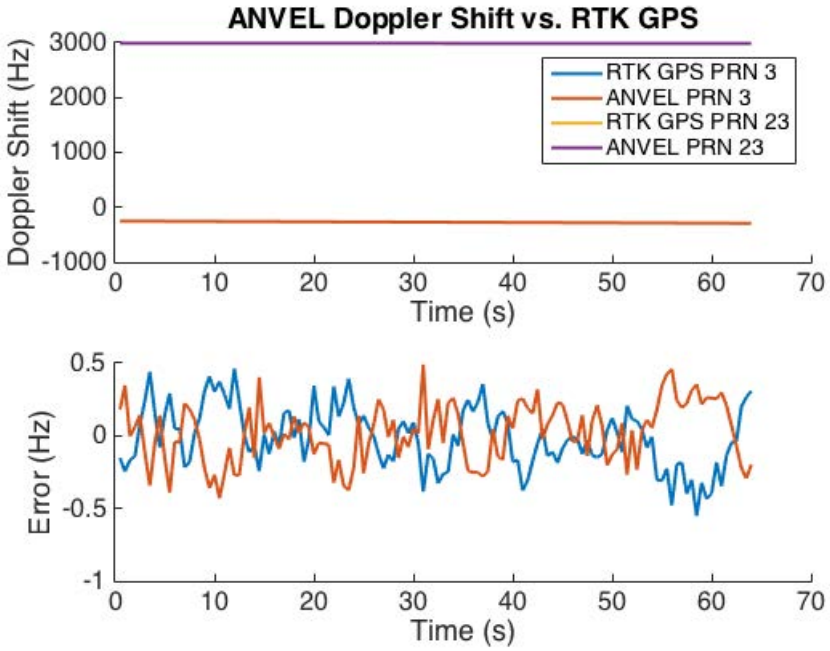

Figure 4.23: GPS Software Module Doppler Shifts Compared to RTK Novatel.

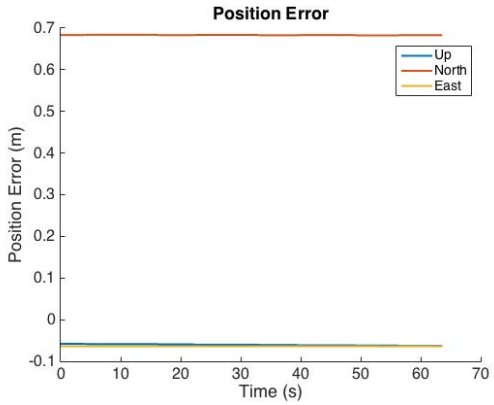

Figure 4.24: GPS Software Limited Position Errors.

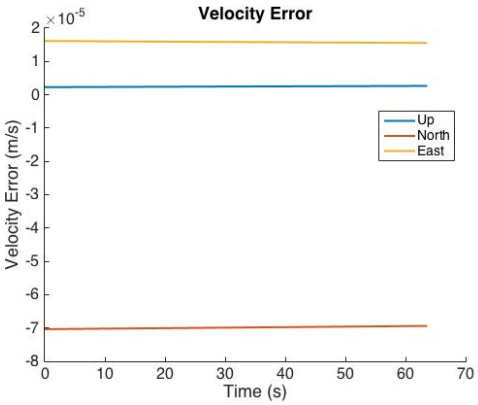

Figure 4.25: GPS Software Limited Velocity Errors.

Ephemeris errors experienced by the standalone Novatel cause the Novatel's position solution to be less accurate than the GPS software module. Also, the Novatel may experience multipath errors and receiver noise that are not easily simulated which cause degradation in position solution. The only errors applied in the above simulation are atmospheric, thus resulting in smaller than expected position and velocity errors. To make the solution from the GPS software module more representative of the Novatel, random noise is added to each pseudorange and position error is added to each satellite position prior to position estimation. Errors are also added to the simulated Doppler shifts and satellite velocities prior to velocity estimation. The magnitude of the additional errors is hand tuned so that the position and velocity errors experienced by the GPS software module is representative of the Novatel receiver shown in Figures 4.26 and 4.27. These errors are very similar to the GPS data provided previously in Figures 4.14 and 4.15 Table 4.4 presents a summation of the RMS errors for the GPS software module and compares them to the standalone Novatel receiver.

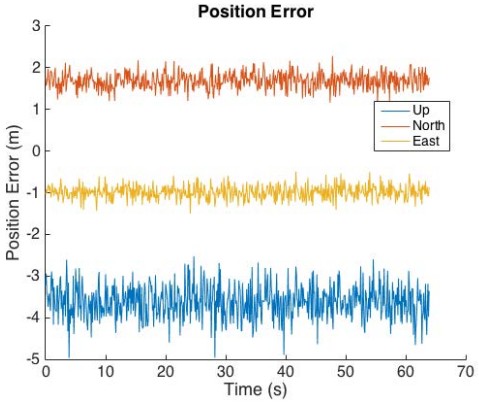

Figure 4.26: GPS Software Position Errors.

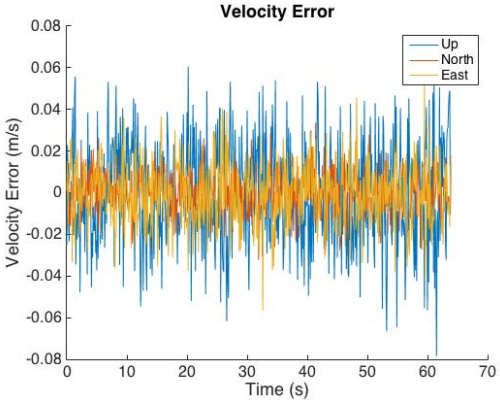

Figure 4.27: GPS Software Velocity Errors.

Table 4.4: GPS Software Module Measurement Errors

Measurement	Novatel RMS Error	GPS Software RMS Error
East Velocity	1.0 cm/s	1.1 cm/s
North Velocity	1.1 cm/s	1.5 cm/s
Up Velocity	2.6 cm/s	2.5 cm/s
East Position	1.0 m	1.0 m
North Position	1.9 m	1.7 m
Up Position	3.8 m	3.7 m
PRN 3 L1 Pseudorange	5.9 m	4.3 m
PRN 3 L1 Doppler Shift	0.29 Hz	0.23 Hz

The statistics for the GPS software module do not agree exactly with statistics for the Novatel due to the fact that many of the errors added in the simulation are random. Furthermore, the goal of this work is not to develop a perfect GPS simulation, but one that is a good approximation for the receiver employed.

Dynamic testing is used to validate that the GPS software module produces an accurate position and velocity solution while the sensor is moving. The dynamic testing data used to validate the GPS software module is gathered by manually driving a vehicle in ANVEL through the downtown Auburn environment with the GPS software module sensor attached to the roof

(as was done with the experimental test shown in Figure 4.8). Position and velocity measurements from ANVEL are used as truth. Figure 4.28 presents an overhead view of position data produced by the GPS software module.

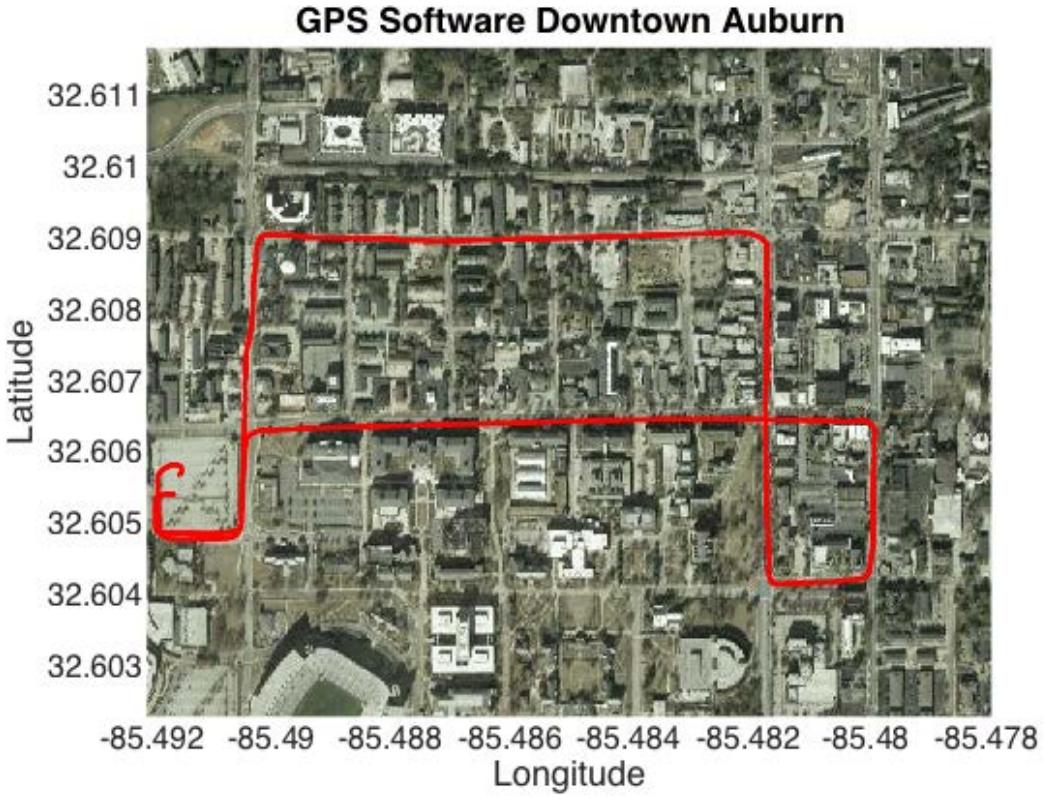

Figure 4.28: GPS Software Downtown Auburn Overhead View.

The GPS software module is able to provide a position solution representative of the real world location simulated by the downtown Auburn environment as the vehicle moves through the environment. Position and velocity errors during the dynamic test are provided in Figures 4.29 and 4.30.

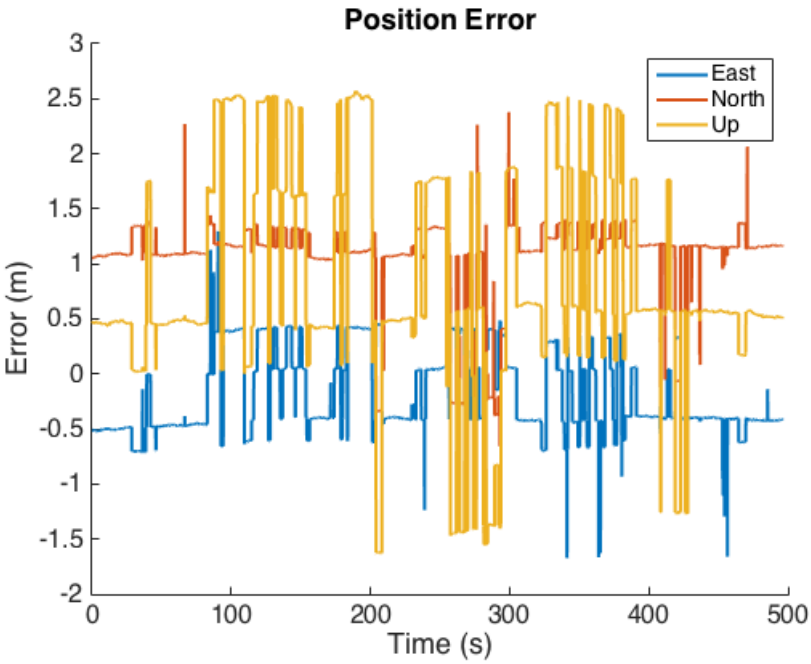

Figure 4.29: GPS Software Downtown Position Error.

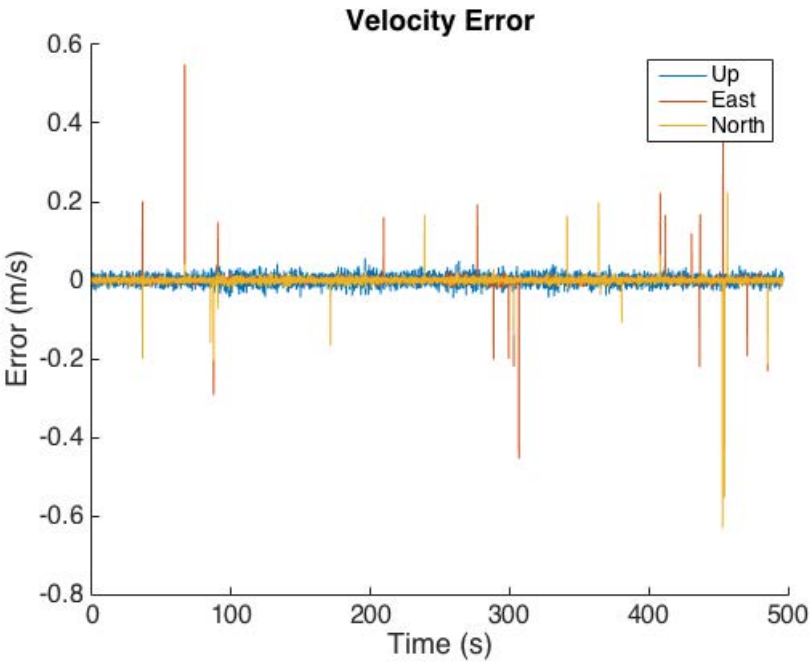

Figure 4.30: GPS Software Downtown Velocity Error.

Note that the only errors added to the GPS simulation during the pictured run are atmospheric errors, thus the low position and velocity errors. Careful examination of Figure 4.29 shows many step changes in position error over the course of the run. These step changes in error are due to satellites being blocked by obstacles in the environment. The number of visible satellites over the course of the test run is shown in Figure 4.31. As seen, the step changes in

position error occur at the same time there is a change in satellite visibility, lending confidence to the fact the ray tracing functionality is performing as desired.

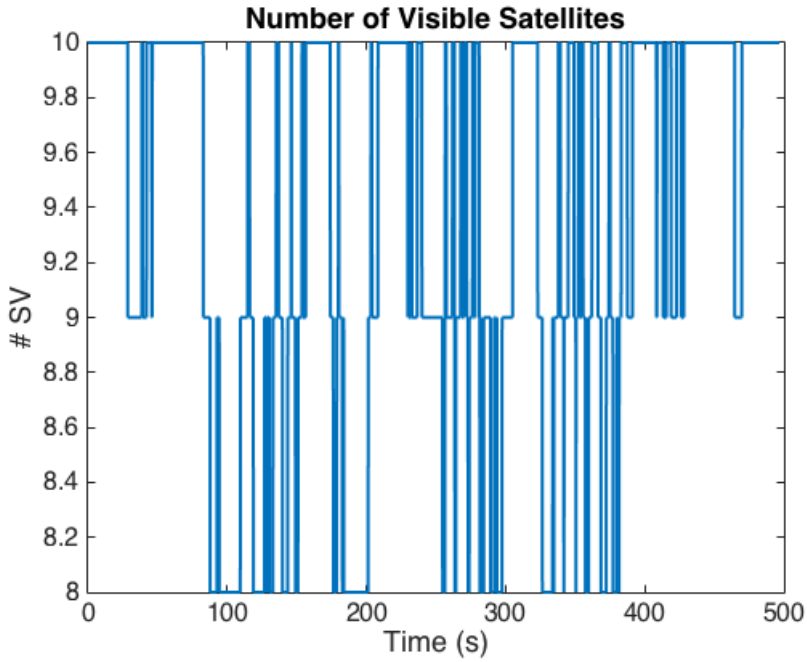

Figure 4.31: GPS Software Downtown Satellite Visibility.

4.3.2 Ray Tracing Software

Testing of the ray tracing section of the GPS module is performed using the 3-D model of the ray tracing test route described in Section 4.7. Experimental data was collected using the GAVLab's G35 with a Novatel receiver recording satellite visibility information. Simulation time is set to begin at the exact time live data collection began to ensure SV positions are the same for the simulation and live testing. The Novatel position is converted to the local ANVEL frame and used to command the position of the vehicle carrying the GPS sensor, ensuring the simulation is representative of the experimental data collection. Figure 4.32 presents satellite visibility data for the simulation and live-sky data.

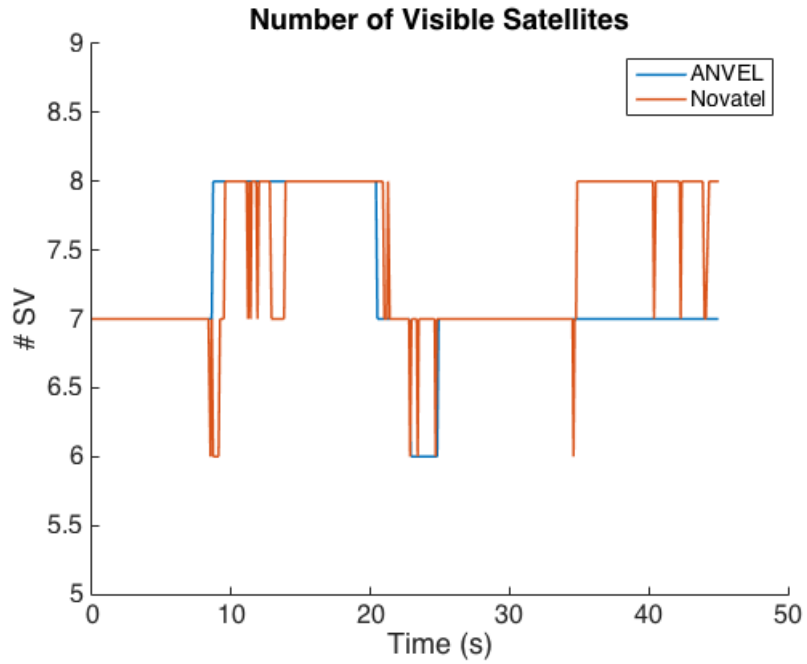

Figure 4.32: GPS Software Ray Tracing Number of Visible Satellites.

As can be seen, both the live data and the simulated data begin with 7 satellites, grow to 8 SV while the vehicle moves from one location to the other, then have 7 satellites visible at the end of the run. Examination of the raw measurements shows the same satellites are going in and out of view for both the live sky and simulated data, shown in Figure 4.33. The variations in visible satellites reported by the Novatel are due to the reflective nature of the test environment and the fact the 3-D model does not exactly match the experimental environment. Signals from blocked and visible satellites are reflected from one or both buildings in the environment causing multipath errors, exhibited by high pseudorange variance reported by the Novatel; however multipath is not modeled by the GPS software module. Also, the GPS software module does not model satellite acquisition, causing small differences in when satellites begin to be used in the solution. Satellite 7 is shown to be blocked by the GPS software module but not the Novatel, likely due to the fact the 3-D model of the test environment does not exactly match the experimental environment. The line of sight for satellite 7 may pass close by the edge of a building that is in a slightly different place in the 3-D model. The trends shown in Figure 4.33 verify the ray tracing portion of the GPS software module is operating as desired.

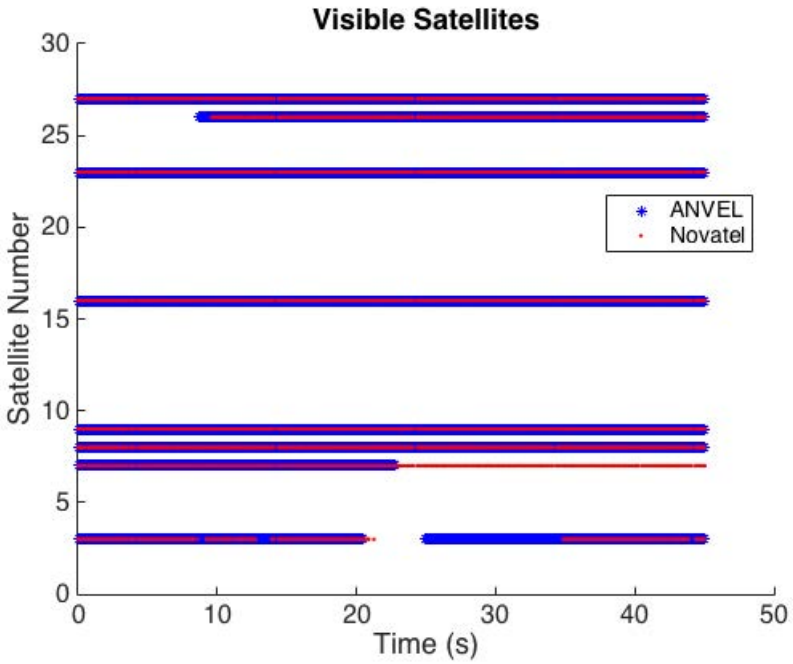

Figure 4.33: GPS Software Ray Tracing Visible Satellites.

4.3.3 IMU Software Module

The first step in validating the IMU software module is to compare the experimental Allan variance results to those obtained from data collected using the IMU software module. Experimental Allan variance results are generated using data collected with a KVH 1725 IMU, described in Section 4.2.2. Generation of the data used for the IMU software module Allan variance data is performed using a ”talker node” to take the place of ANVEL during model validation. The talker node simply publishes IMU data for the IMU software module to use which is representative of a static IMU on a level surface with no errors. The IMU software module receives perfect IMU data from the talker node, then corrupts and publishes the corrupted data until 24 hours worth of messages are published. The same code used to generate the Allan variance results for the experimental data is used to generate results for the IMU software module. A comparison of the Allan deviation plots for the KVH 1725 IMU and the IMU software module Z gryoscopes can be seen in Figure 4.34.

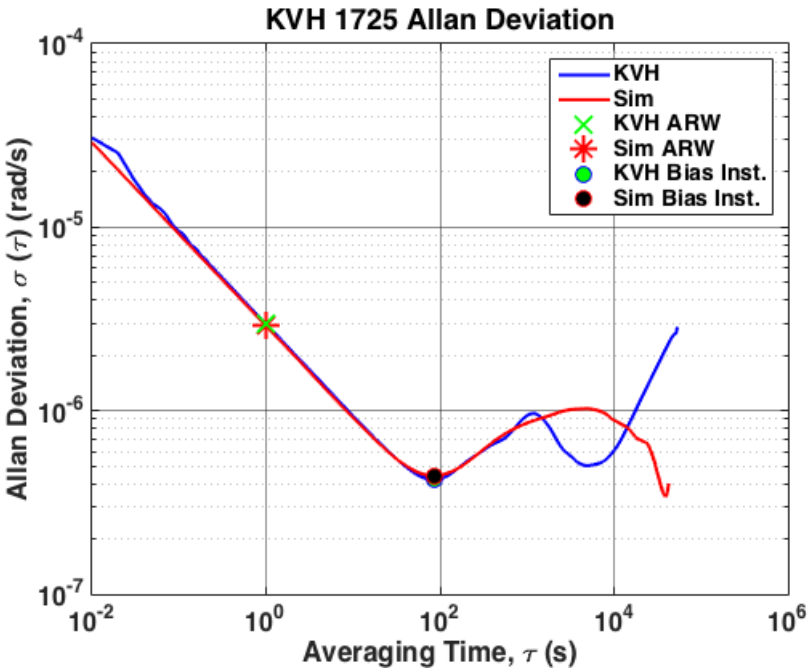

Figure 4.34: Comparison of Allan Deviation for IMU Software Module.

Examining the Allan deviation plots shows that the IMU software module accurately represents the ARW exhibited by the KVH. The ARW exhibited by the IMU software module is 2.99×10^{-6} rad/s/$\sqrt{\text{Hz}}$ as compared to 2.96×10^{-6} rad/s/$\sqrt{\text{Hz}}$ from the KVH. The bias instability for the IMU module is 4.37×10^{-7} rad/s, which is also close to the 4.21×10^{-7} rad/s exhibited by the KVH. Examination of the Allan deviation shows the two models exhibit similar behaviors until the averaging time reaches around 1,000 seconds and exhibit a slope of +1/2 in the same area of the plot, showing the Markov process is modeled accurately. The discrepancies shown between then slopes of the simulated and KVH Allan deviations are due to the fact that the experimental data from the KVH contains error sources that are not modeled in the simple models employed. The purpose of this work is not to provide an exact error model, but a good base for future researchers to expand upon. The fact that the ARW, bias instability, and shape of the Allan deviation plot match closely shows the developed models provide a good approximation of the KVH. A final tabulation of error characteristics for the IMU software and hardware modules is presented in Tables 4.6 and 4.7 in Section 4.4.3.

As stated in Section 4.2.2, experimental identification of the Markov process is proven to be quite difficult. When tuning the error simulations, the author found the error characteristics obtained from the autocorrelation process using KVH data to be inadequate for use in the

employed models. Only by hand tuning the Markov characteristics and analyzing many data sets was the author able to obtain the Allan deviation results shown. The final value used for σ_b^2 is 1.8×10^{-11} rad/s^2 with a time constant of 10,000 seconds.

As an additional check, data from the IMU software module is compared to ANVEL IMU data during a dynamic test run. The only difference between the ANVEL data and the IMU software module should be the noise and bias added by the IMU software module. Figure 4.35 shows Z gyroscope measurements during a test scenario where the vehicle was manually driven through the downtown Auburn test environment while Figure 4.36 shows the error. The outputs from the IMU software module are differenced by the ANVEL truth measurements to examine the errors added by the IMU software module.

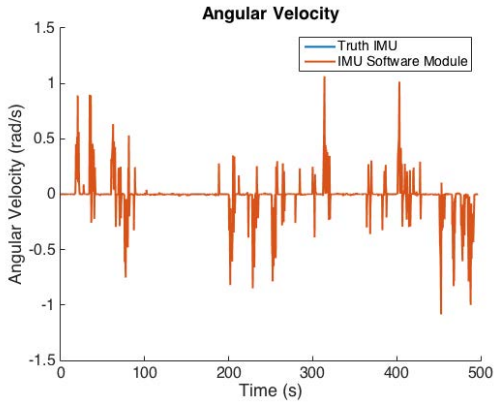

Figure 4.35: IMU Software Yaw Rate in Downtown Auburn.

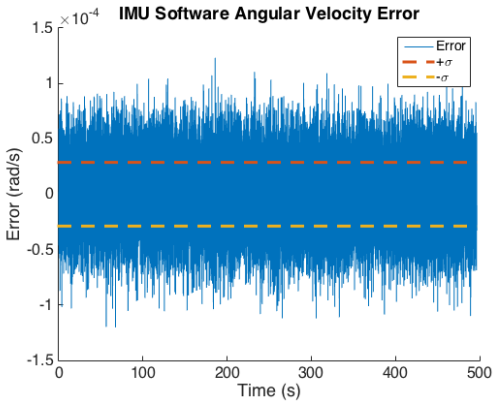

Figure 4.36: IMU Software Yaw Rate Error.

The standard deviation of the IMU error in the run pictured is 2.9476×10^{-5} rad/s, which agrees with the ARW value obtained previously when accounting for the 100 Hz sampling rate. The close match of the error statistics added by the IMU software module and the IMU characterization results show the IMU software module is adding errors representative of the hardware IMU.

When examining the dynamic data, a time delay between the time stamps of the truth IMU and the IMU software module was discovered. A plot of the time delay is shown in Figure 4.37.

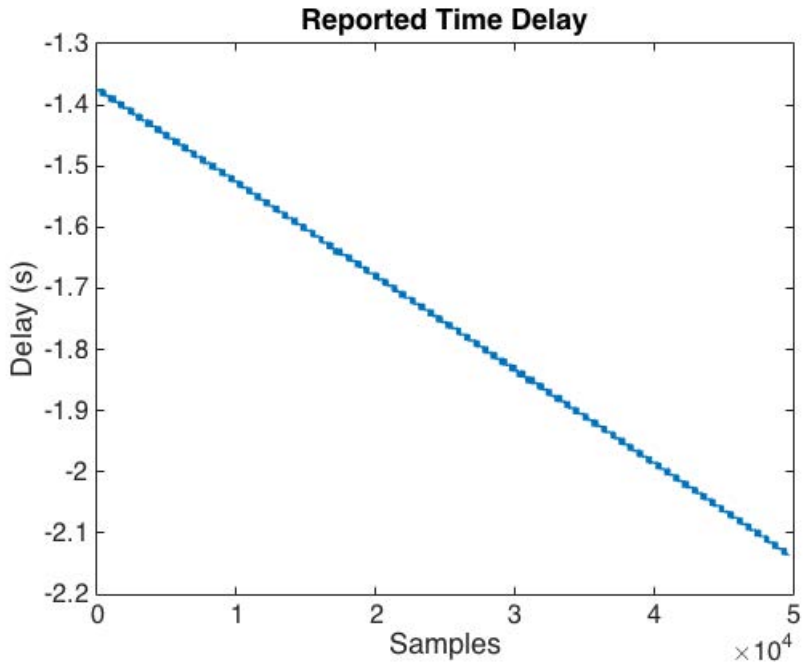

Figure 4.37: Changing Time Delay Between ANVEL IMU and IMU Software Module.

The time delay is observed to decrease over time, suggesting the software module is running more slowly than the data is being provided to it. Additionally, the time bias is negative, suggesting the IMU software module is time stamping data before that data is even generated. Looking at the average time between time stamps reveals the IMU software module is operating at exactly 100 Hz, while the truth IMU data is stamped at a rate of around 100.1 Hz. The IMU software module only has a buffer of 10 messages to prevent data from accumulating. Operating at 0.1 Hz slower means that every 10 seconds, one more message is produced than the software module is processing. With a buffer size of only 10 messages, it only takes 100 seconds for more than 10 messages to be in the buffer causing the IMU software module to lose a message. A missed message would cause the time bias to decrease by 0.01 s, which it does many times. However, examination of the data set shows the recorded messages for the truth IMU and software module are the same length, implying the software module did not miss any messages. If the step change in delay is due to missed messages, that would indicate the IMU software module lost around 100 messages over the course of the run, which does not agree with the run lengths being equivalent. Examining IMU data with plotted against epochs instead of time in Figure 4.38 demonstrates that no messages are lost by showing the two signals are aligned at the end of the test run.

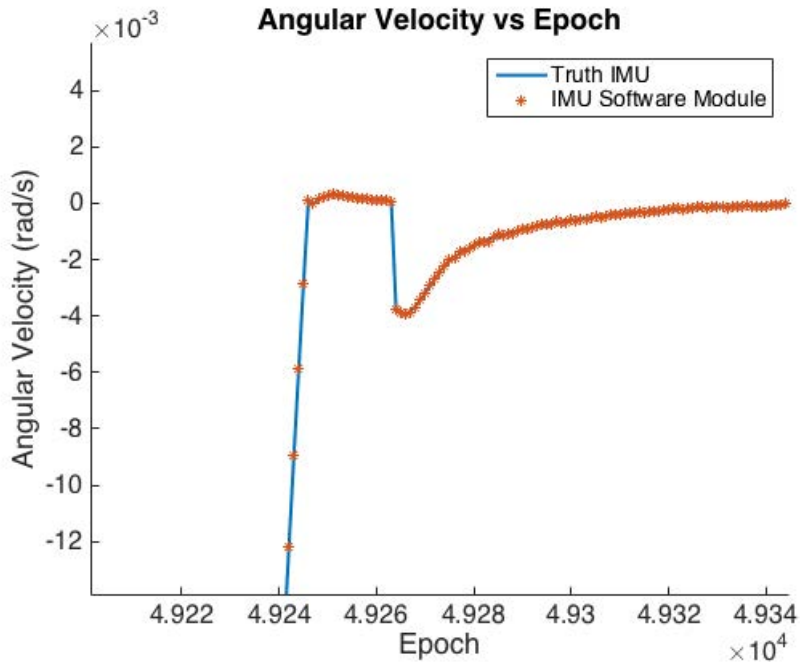

Figure 4.38: Zoomed in View of ANVEL IMU and IMU Software Signals vs. Epoch.

The likely cause of of this issue is the time stamping method employed in the ROS gateway provided by TARDEC. Figure 4.39 shows the instability of the timing. The gateway converts a time stamp from the ANVEL PC to ROS time while the IMU software module simply uses ROS time to stamp the messages. Examining Figure 4.39 shows that the time stamps of the ROS gateway vary while the IMU software module time stamps are much more stable. This indicates that the method employed in the gateway is losing precision when performing time conversions likely causing the timing phenomenon observed. The time drift shown in Figure 4.37 may be explained by the two computer clocks drifting at different rates. Using a common clock between the ANVEL PC and the Linux PC as well as changing the time stamping method within the ROS gateway provided by TARDEC are potential solutions for the timing issues. This timing issue is not observed in the GPS modules due to the fact the GPS modules use the Linux PC for all time stamping.

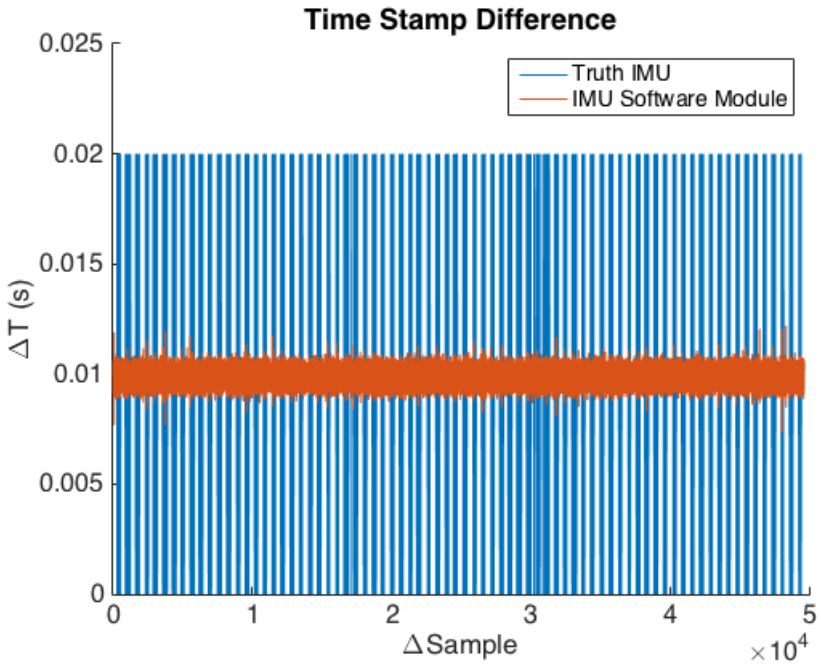

Figure 4.39: Difference in Time Stamps Between Samples.

4.3.4 Wheel Speed Sensor Software Module

Testing the output of the WSS software module is a relatively simple process. Because the WSS module only adds error due to quantization, the output should never be more than a full quantization step away from ANVEL wheel speeds. For an initial comparison, data is collected with a vehicle traveling at a constant speed in the large parking lot in ANVEL with the WSS software module running. Wheel speed reported by ANVEL is compared to the wheel speeds calculated from the WSS software module in Figure 4.40 for a 10 mph run. The maximum error between ANVEL wheel speeds and WSS wheel speeds in the figure is 0.102 m/s, which is very close to one quantization step.

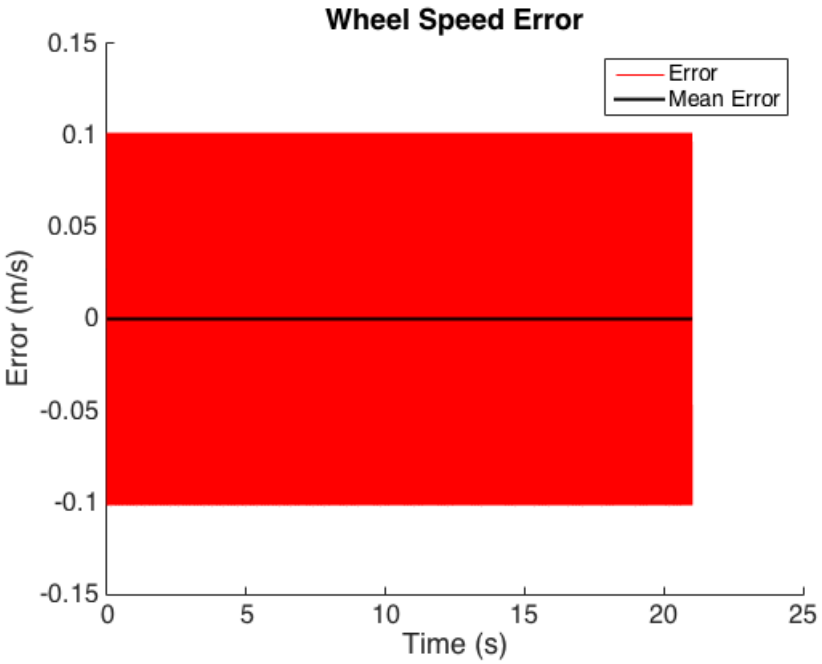

Figure 4.40: Wheel Speed Software Module Error at 10 mph.

Figure 4.40 represents the best case scenario when only the WSS software module is running on the Linux PC. In reality, the WSS software module will be only one of many nodes running at one time. Having multiple nodes running introduces more clock jitter when sampling, yielding different error characteristics for the WSS. Figures 4.41 and 4.42 show the WSS software module error with increased processor load.

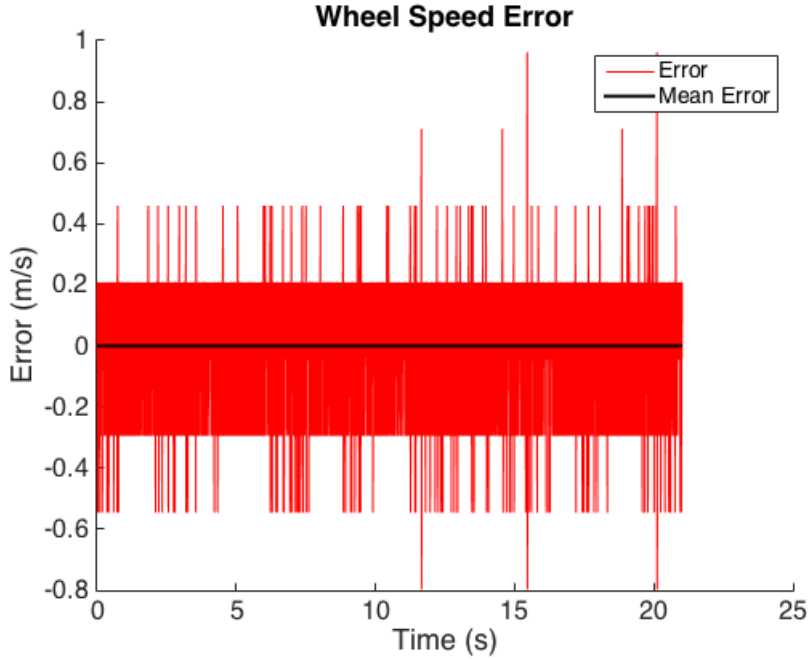

Figure 4.41: Wheel Speed Software Module Error with Increased Processor Load.

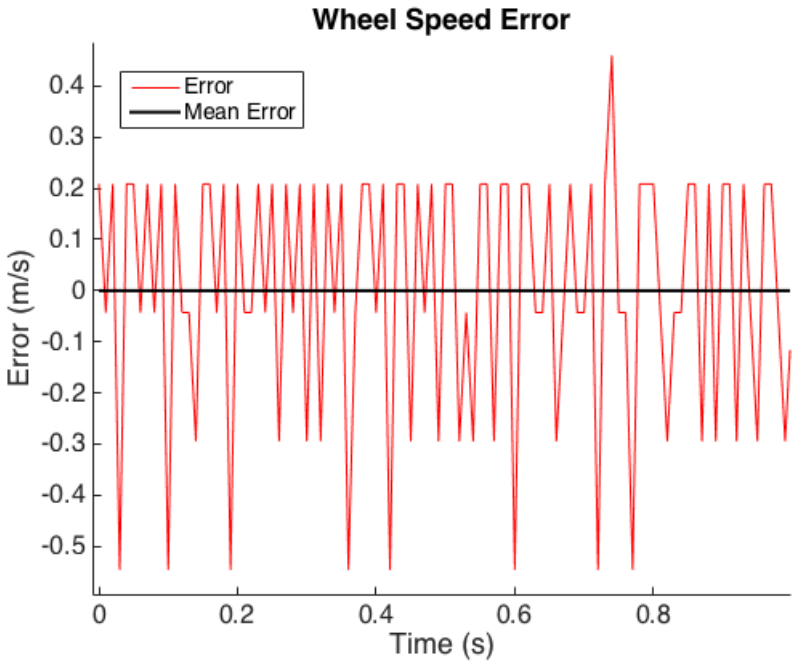

Figure 4.42: Close View of Wheel Speed Software Module Error.

The behavior shown in Figures 4.41 and 4.42 matches closely with that observed from the hardware WSS in Section 4.2.3. Figure 4.43 compares the standard deviation of the delta encoder counts for wheel speed data collected using the hardware WSS and the WSS software module. The WSS software module standard deviations are observed to grow with increasing speed with slope similar to that of the hardware WSS, showing the WSS module accurately captures the error dynamics of the hardware WSS. Because more nodes were running while collecting the WSS software data, the differences in slope may be attributed to increased clock jitter due to increased processor load. The WSS software module also experiences the shifting time bias discussed in Section 4.3.3, suggesting the timing issue is present in all modules using data passed in the ROS gateway provided by TARDEC.

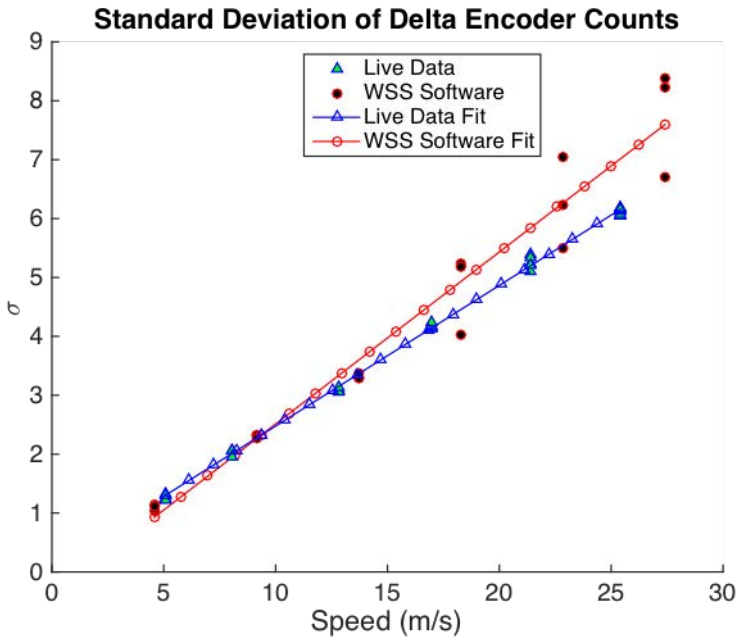

Figure 4.43: WSS Software Module Standard Deviation of Encoder Counts.

As described in Section 4.2.3, the wheel speed error for the experimental setup was observed to grow as speed increases. This trend is also observed in the WSS software module, but in the opposite direction. Figure 4.44 presents a comparison of the growing wheel speed errors for the WSS software module and experimental data. The difference in error trends for the two data sets is possibly due to a changing wheel radius as speed increases, or possibly the incorrect estimation of the experimental wheel radius. Changing the experimental wheel radius used in calculation of wheel speeds yields the comparison shown in Figure 4.45. The experimental wheel radius used to produce Figure 4.45 is 0.332 m, compared to the estimated radius of .323 m, yielding a much closer comparison between the two error trends. As shown, changing the radius by only 2.8% has a large effect on the error trends of the WSS and is a likely cause of the difference between the simulated and experimental error trends.

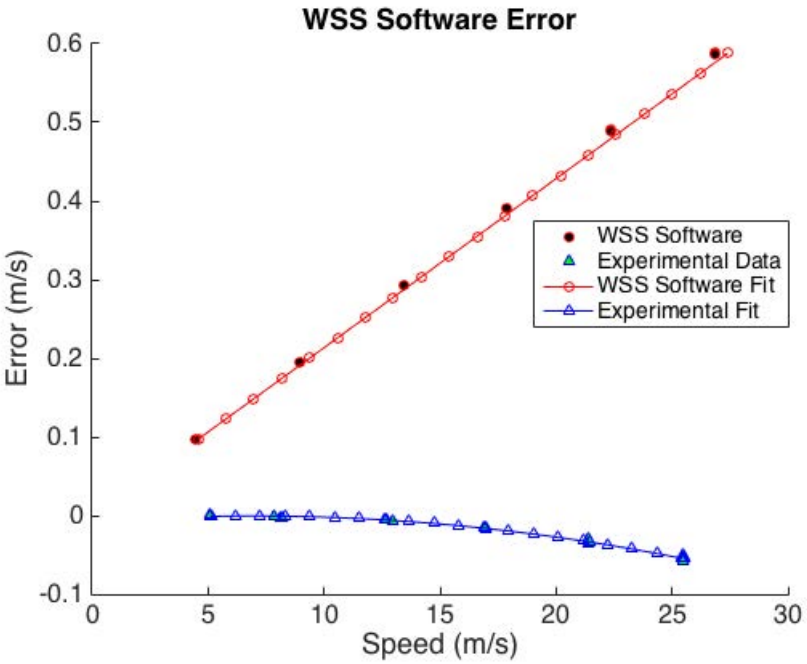

Figure 4.44: WSS Software Module Wheel Speed Error.

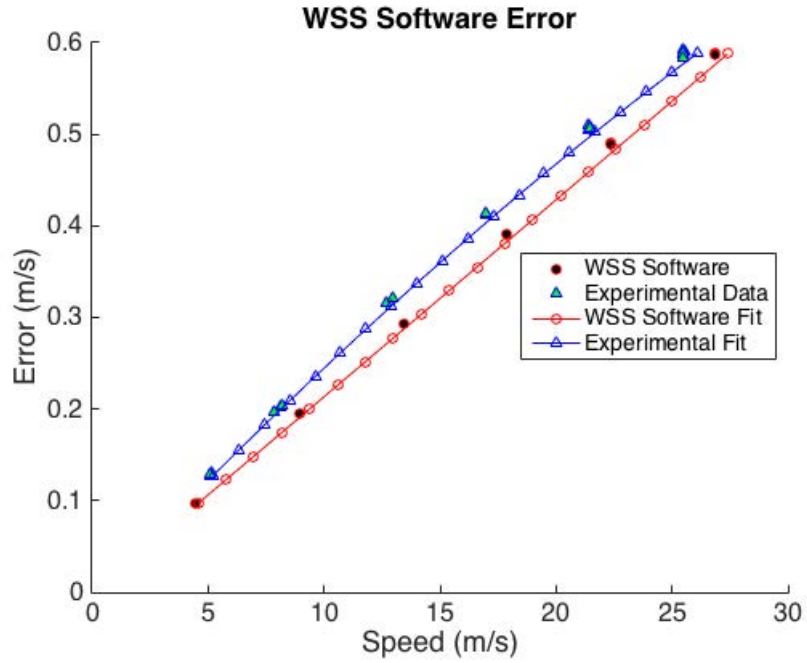

Figure 4.45: WSS Software Module Wheel Speed Error with Modified Radius.

4.4 Hardware Module Validation

The validation process for the hardware modules is very similar to the validation process for the software modules. The same data sets and test environments used for validation of the software modules in Section 4.3 are now used for the hardware modules. Data from each of the

hardware modules for validation is captured using the same sensor drivers as used by the real sensors to ensure the hardware modules may be sampled as a real sensor. Again, the hardware sensor characterization data found in Section 4.2 is used as a baseline for module development and validation.

4.4.1 GPS Hardware Module

As with the software module, the first step in validation for the GPS hardware module is simulating a static antenna situated on the roof of the Woltosz building. However, the GPS measurements evaluated for the hardware module are now evaluated from the output by a Novatel ProPak V-3 receiving RF signals output by the Spectracom simulator. The Novatel is sampled using the same C++ ROS driver as the GPS receiver in the GAVLab navigation unit. A visualization of the test setup was provided in Section 4.1.2. Truth for position and velocity are the known position and velocities of the static antenna, while pseudorange and Doppler truth is obtained from live-sky Novatel data with RTK corrections. The Spectracom is provided with satellite ephemeris and ionospheric delay data for the time period simulated.

Examination of the pseudoranges for SV 3 and 14 reveal large differences in pseudoranges between the simulated and live-sky data, shown in Figures 4.46 and 4.47. Large differences are also observed in the Doppler shift measurements reported by the two receivers in Figures 4.48 and 4.49. The pseudoranges exhibit growing errors over time while the Doppler measurements exhibit a bias. These errors are thought to be the result of changes in receiver clock drift between the live-sky run and the simulation as the two runs were conducted months apart.

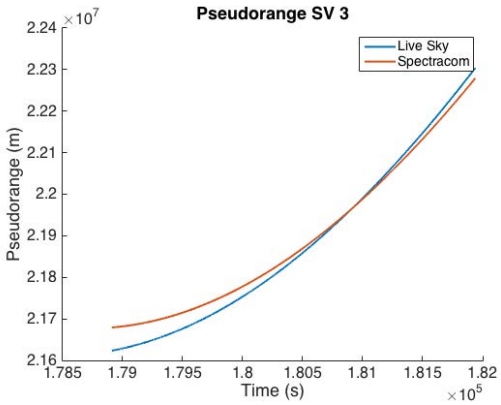

Figure 4.46: GPS Hardware Pseudorange Errors: SV 3.

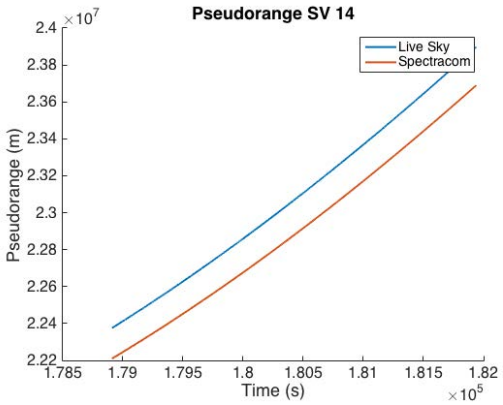

Figure 4.47: GPS Hardware Pseudorange Errors: SV 14.

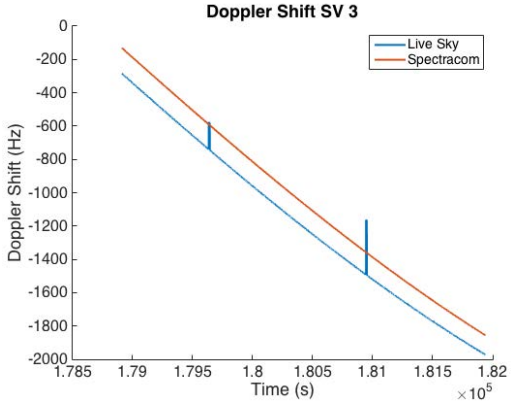

Figure 4.48: GPS Hardware Doppler Shift Errors: SV 3.

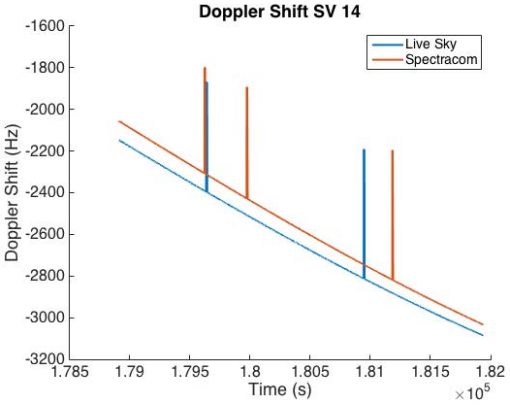

Figure 4.49: PS Hardware Doppler Shift Errors: SV 14.

The errors in pseudorange and Doppler shift are not realized in the position and velocity data shown in Figures 4.50 and 4.51. The position exhibits changing errors but settles at around 750 seconds with a maximum of 3.5 m in the up direction. Error characteristics for GPS hardware position and velocity are presented in Table 4.5. Comparing these error characteristics to the errors experienced by the standalone Novatel in live-sky data in Table 4.5 show that the GPS hardware module outputs a position solution with less RMS error than a standalone Novatel in live-sky conditions. The velocity solution is representative of a standalone Novatel. The errors observed in the GPS data produced using the Spectracom are only due to the Spectracom itself; the position and velocity commanded to the Spectracom are the true position and velocity from ANVEL with no errors added. To make the position errors closer to experimental data, errors may be manually added to the positions prior to Spectracom commands; however, the magnitude of these errors would need to be tuned so the error characteristics match experimental data.

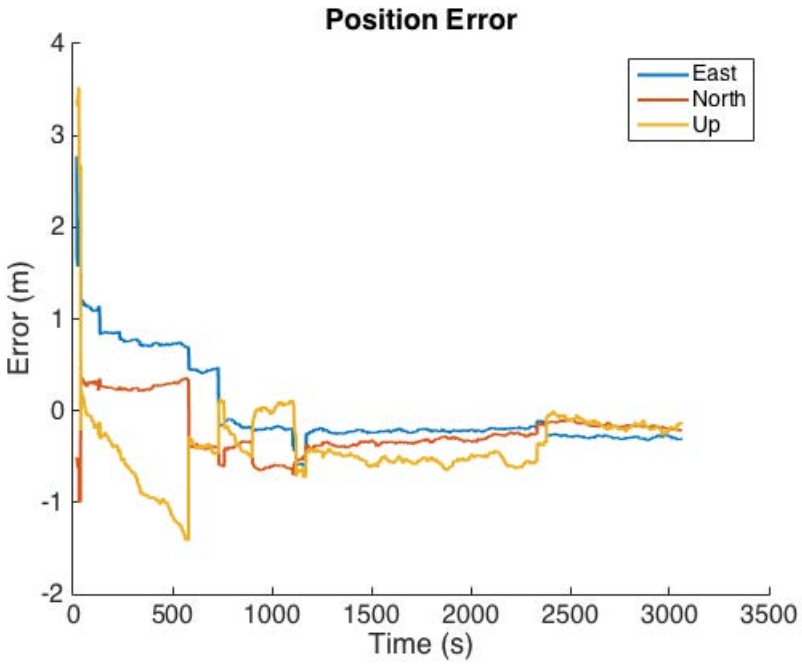

Figure 4.50: GPS Hardware Static Position Error.

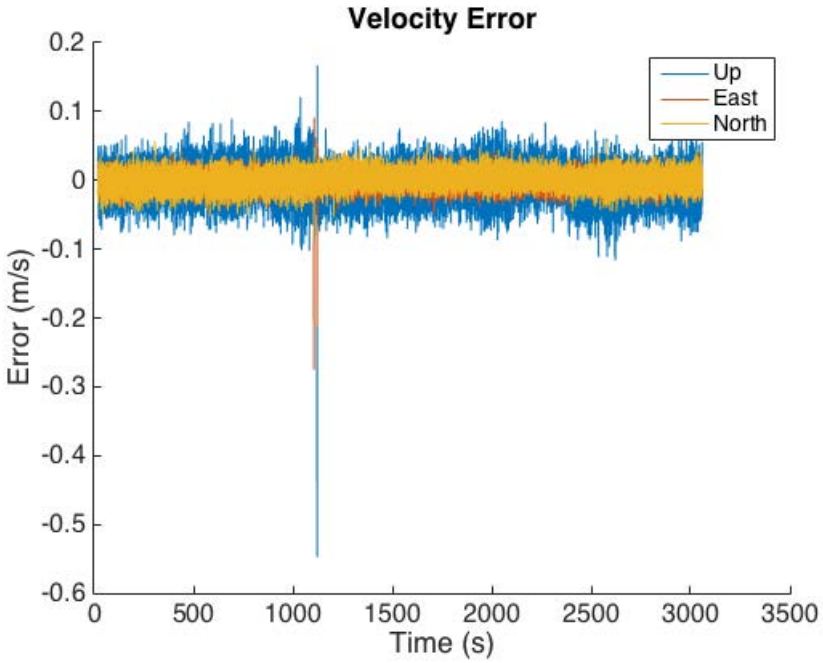

Figure 4.51: GPS Hardware Static Velocity Error.

Table 4.5: GPS Hardware Module Measurement Errors

Measurement	Novatel RMS Error	GPS Software RMS Error
East Velocity	1.0 cm/s	1.1 cm/s
North Velocity	1.1 cm/s	1.2 cm/s
Up Velocity	2.6 cm/s	2.4 cm/s
East Position	1.0 m	0.4 m
North Position	1.9 m	0.3 m
Up Position	3.8 m	0.6 m

As with the GPS software module, dynamic testing of the GPS hardware module is performed by driving a vehicle through the downtown Auburn environment in ANVEL and comparing position and velocity outputs to ANVEL truth. Collection of valid dynamic data is more difficult than the GPS software module due to the fact the vehicle must remain stationary for at least 30 seconds at the beginning of the run so the Novatel receiver can acquire the simulated satellite signals. High dynamics also tend to cause the Novatel to lose tracking of multiple satellites. Figures 4.52 and 4.53 represent the position and velocity error of the GPS hardware module as compared to ANVEL truth.

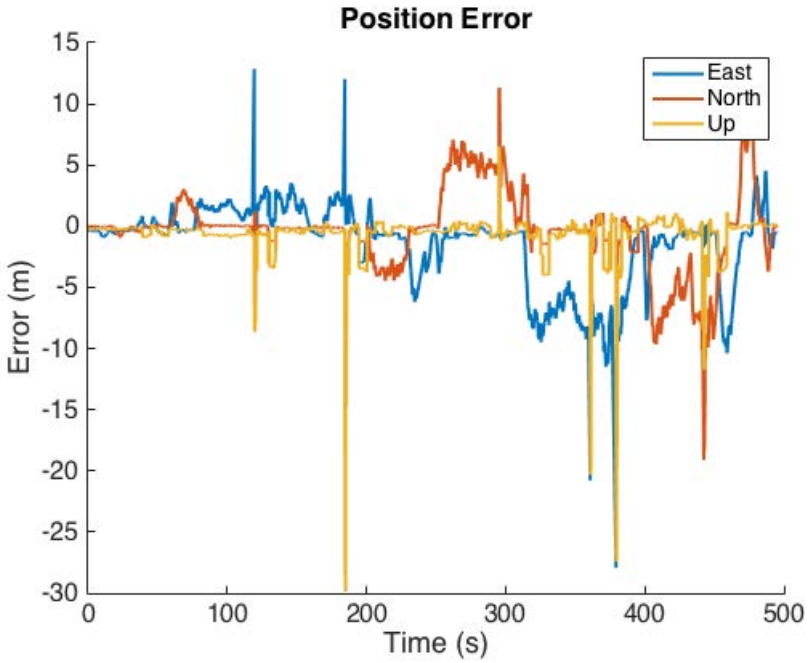

Figure 4.52: GPS Hardware Dynamic Position Error.

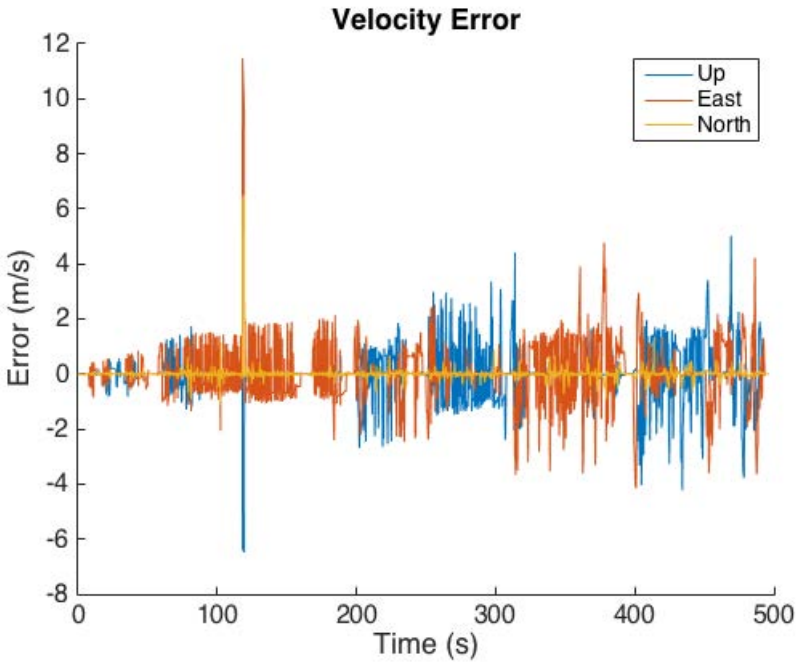

Figure 4.53: GPS Hardware Dynamic Velocity Error.

Large position and velocity errors are observed that appear to grow over the course of the run. These errors are thought to be caused by the time delay inherent to the Spectracom due to the fact that when plotted on a map of the environment, the position solution follows the test route, shown in Figure 4.54.

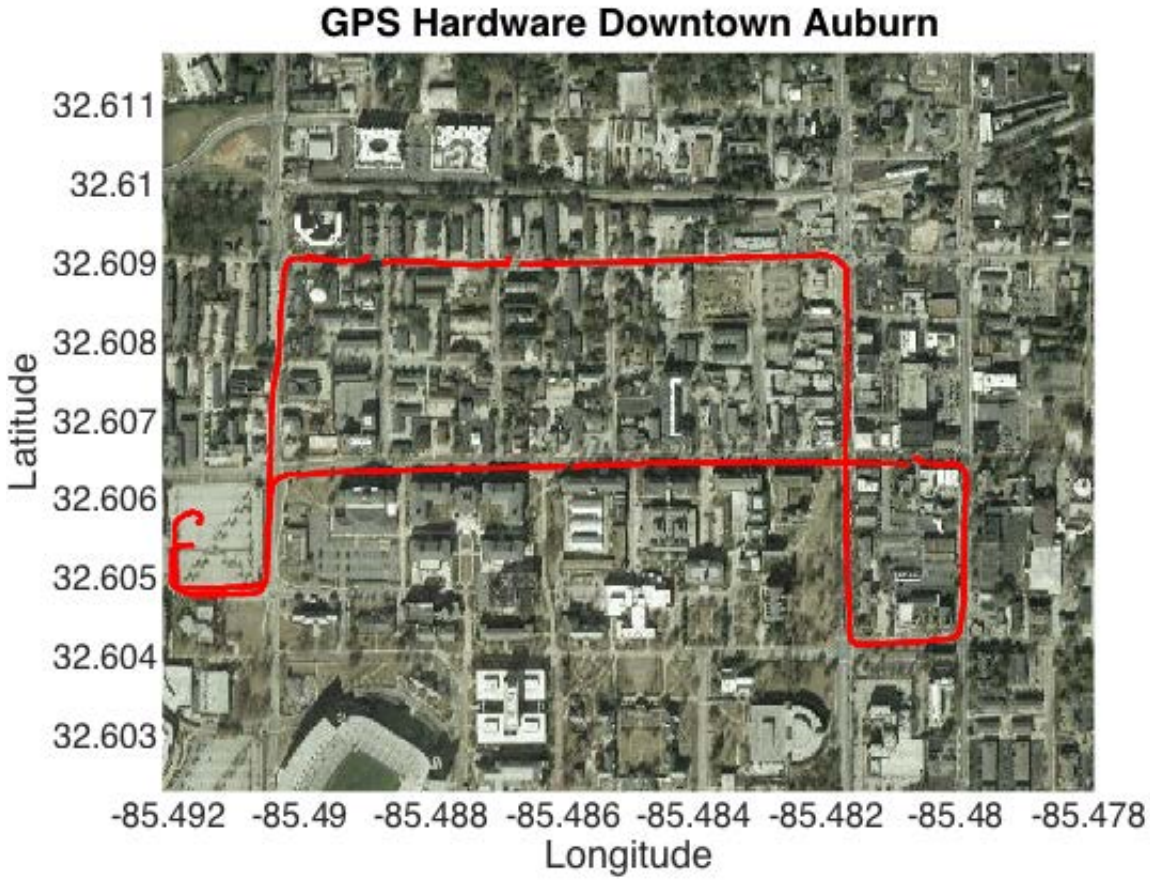

Figure 4.54: GPS Hardware Downtown Auburn Overhead View.

Attempts to align the position and velocity data in time proved to be non-trivial. Shifting the measurements by the known time bias produces errors as shown, but still produces higher than expected errors in the North and East directions. The Up direction error is most unaffected due to the fact the test route contains little variation in elevation. The large East and North errors suggest the timing issues related to the Spectracom within the GPS hardware module are not only due to the known bias, but include some other source of timing error. The errors shown typically only grow in either the East or North direction, but not both at the same time. This is due to the fact that along the test route, the vehicle is typically only traveling in the East or North direction. As the vehicle travels North, the East position is roughly constant while the North position is changing. The time delay causes the Spectracom to output an old position, causing North position error to grow as the vehicle travels. When the vehicle turns and travels in the East direction, the same phenomenon is observed in the North position error.

The magnitudes of these errors appear to grow with time as well, suggesting the time bias in the Spectracom is growing throughout the course of the run.

As with the GPS software module, step changes in Up position error can be observed over the course of the test run. These step changes are also due to satellites moving in and out of view. Satellite visibility data for the downtown Auburn test route is shown in Figure 4.55. The step changes in position error occur at the same time there is a change in satellite visibility, lending confidence to the fact the ray tracing functionality is performing as desired.

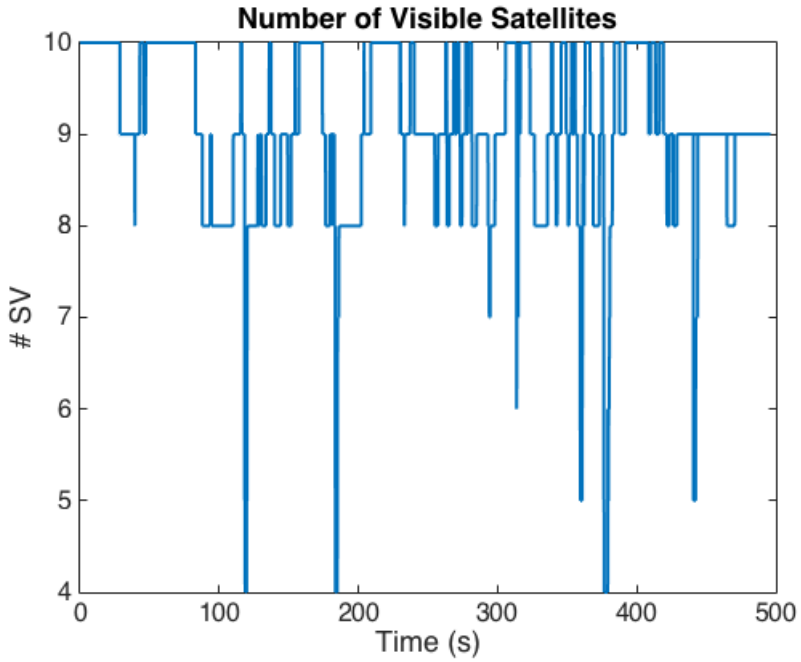

Figure 4.55: GPS Software Downtown Satellite Visibility.

4.4.2 Ray Tracing Hardware

Testing of the ray tracing section of the GPS hardare module module is performed using the same test environment as in the GPS software module. The goal in testing the ray tracing portion of the GPS modules using the Spectracom is to ensure that setting blocked SV power to the minimum value causes the Novatel to lose tracking on that SV and not use it for a PVT solution. Simulation time is set to begin at the exact time data collection began during live data collection to ensure SV positions are the same for the simulation and live testing. The vehicle is simulated driving through the 3-D model in ANVEL using positions reported by the Novatel

during experimental data collection. SV visibility is recorded for the simulation and compared to live testing in Figure 4.56.

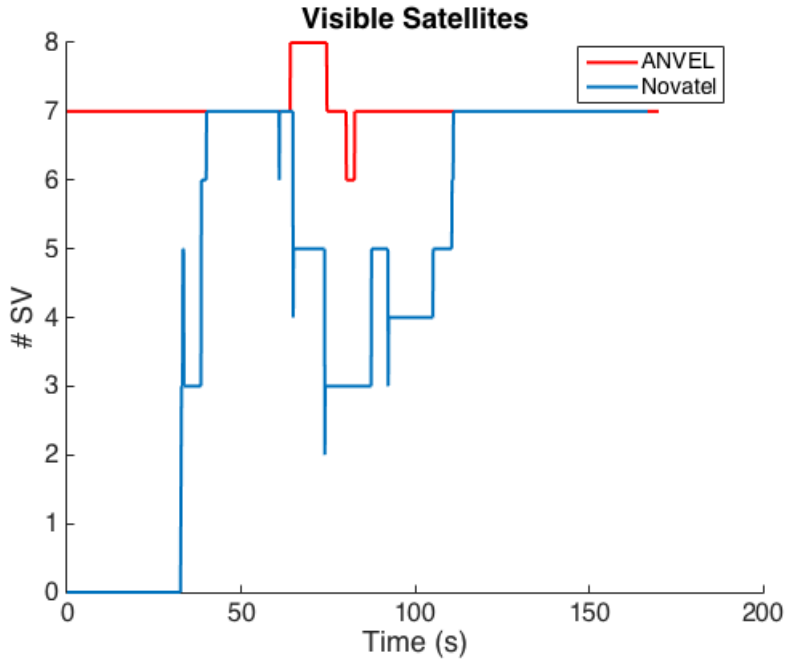

Figure 4.56: GPS Hardware Satellite Visibility.

The satellite visibility results shown demonstrate the difficulties in testing with the GPS hardware module. The positions commanded to the Spectracom are derived from Novatel data recorded in live sky which contain small jumps in position between epochs. These small jumps make it difficult for the Novatel used in testing to acquire satellite signals produced by the Spectracom and maintain lock while moving, explaining the discrepancies at the start of the run and during movement in the middle. The fact the truth data and GPS hardware data report the same number of visible satellites before and after movement give some confidence the ray tracing portion of the GPS hardware module is performing as desired. Also, close examination of GPS hardware dynamic data through downtown Auburn in Figure 4.54 reveals step changes in up position error similar to the GPS software module caused by satellite outages, giving more confidence the ray tracing portion of the GPS hardware module is operating correctly.

4.4.3 IMU Hardware Module

The IMU hardware module uses the same error model characteristics in generation as the IMU software module, so the error characteristics should be very similar. The main validation

required for the IMU hardware module is ensuring the packaging and serial transmission of the IMU data is compatible with the driver used to sample the actual KVH and does not effect the error characteristics of the model. To verify this, the talker node used in verification of the IMU software module is used to provide the IMU hardware module with perfect IMU data to corrupt. The IMU hardware module corrupts the inertial data and outputs measurements as a RS-422 signal. The serial output is then sampled for 24 hours by the same ROS node used to sample the KVH IMU and an Allan deviation plot is produced, shown in Figure 4.57.

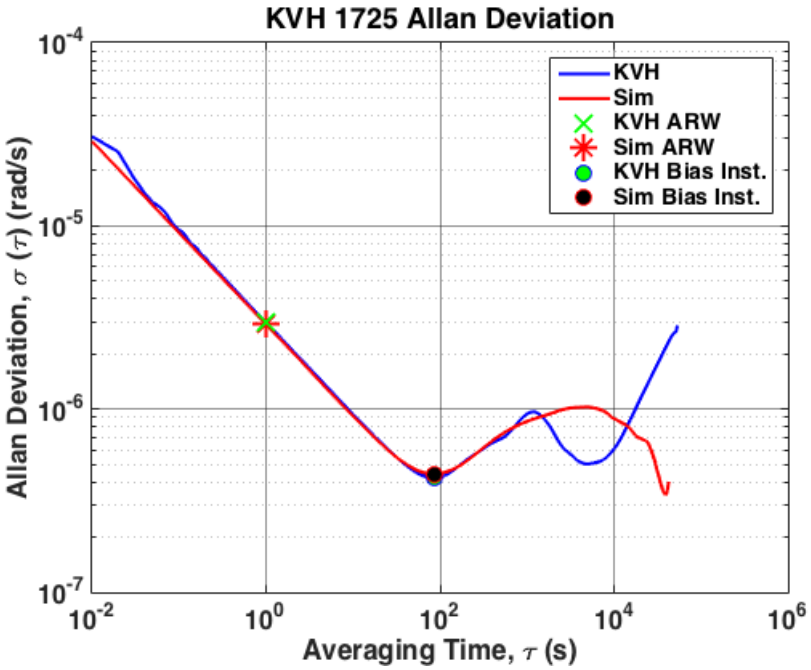

Figure 4.57: Comparison of Allan Deviation for IMU Hardware Module.

The ARW exhibited by the IMU hardware module is 2.96×10^{-6} rad/s/$\sqrt{\text{Hz}}$ as compared to 2.96×10^{-6} rad/s/$\sqrt{\text{Hz}}$ from the KVH. The bias instability for the IMU module is 4.43×10^{-7} rad/s, which is also close to the 4.22×10^{-7} rad/s exhibited by the KVH. Again, the discrepancies shown between the slopes of the simulated and KVH Allan deviations are due to the fact that the experimental data from the KVH contains error sources that are not modeled in the simple models employed. A final tabulation of the error characteristics of the IMU software and hardware modules as compared to experimental data and manufacturer specifications is presented in Tables 4.6 and 4.7. The close match of error characteristics between the developed modules and the KVH 1725 show that the simple models employed provide a good approximation of the true sensor behavior.

Table 4.6: Comparison of KVH and Simulated Accelerometer Error Characteristics

Parameter	Axis	Manufacturer Spec.	Experimental Value	IMU Software	IMU Hardware
Velocity Random Walk ($m/s^2/\sqrt{Hz}$)	X	1.09×10^{-3}	7.38×10^{-4}	7.38×10^{-4}	8.15×10^{-4}
	Y	1.09×10^{-3}	8.35×10^{-4}	8.33×10^{-4}	8.33×10^{-4}
	Z	1.09×10^{-3}	8.04×10^{-4}	8.04×10^{-4}	8.02×10^{-4}
Bias Instability (m/s^2)	X	9.09×10^{-4}	3.72×10^{-4}	1.30×10^{-4}	1.37×10^{-4}
	Y	9.09×10^{-4}	4.54×10^{-4}	1.40×10^{-4}	1.38×10^{-4}
	Z	9.09×10^{-4}	3.91×10^{-4}	1.34×10^{-4}	1.40×10^{-4}

Table 4.7: Comparison of KVH and Simulated Gyroscope Error Characteristics

Parameter	Axis	Manufacturer Spec.	Experimental Value	IMU Software	IMU Hardware
Angular Random Walk ($rad/s/\sqrt{Hz}$)	X	4.84×10^{-6}	2.96×10^{-6}	2.96×10^{-6}	2.96×10^{-6}
	Y	4.84×10^{-6}	2.89×10^{-6}	2.89×10^{-6}	2.90×10^{-6}
	Z	4.84×10^{-6}	2.96×10^{-6}	2.96×10^{-6}	2.96×10^{-6}
Bias Instability (rad/s)	X	4.84×10^{-7}	4.21×10^{-7}	4.61×10^{-7}	4.49×10^{-7}
	Y	4.84×10^{-7}	4.22×10^{-7}	4.40×10^{-7}	4.43×10^{-7}
	Z	4.84×10^{-7}	3.59×10^{-7}	4.51×10^{-7}	4.40×10^{-7}

To test during dynamic scenarios, outputs from the IMU hardware module are differenced by the ANVEL truth measurements during test runs in the downtown Auburn environment. Figure 4.58 shows the entirety of the run while Figure 4.59 shows the error added by the IMU hardware module. The standard deviation of the error shown in Figure 4.59 is 2.9492×10^{-5} rad/s, agreeing with the value found previously for ARW when accounting for the 100 Hz sampling rate.

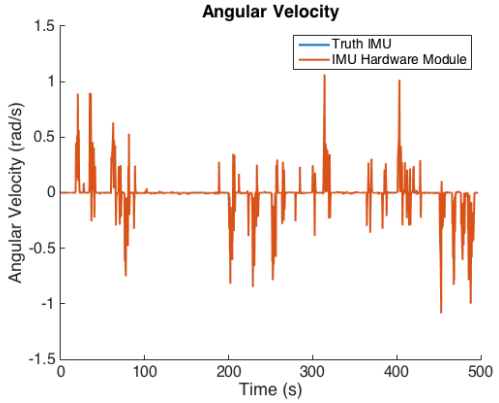

Figure 4.58: IMU Hardware Yaw Rate in Downtown Auburn.

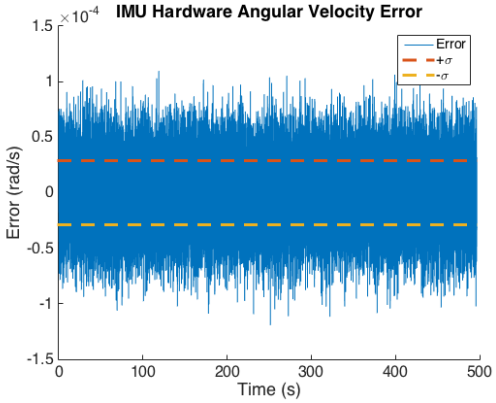

Figure 4.59: IMU Hardware Yaw Rate Error.

The timing issues between ANVEL truth measurements and module output data described in Section 4.3.3 is also observed in Figure 4.60 for the IMU hardware module. The time delay is negative, again suggesting the IMU module is stamping data before the data is output by ANVEL. Again, the two data sets are the same length and the IMU hardware module obviously cannot time stamp data before it is even published by the gateway. This observation also suggests a time stamping issue which must be addressed as discussed previously.

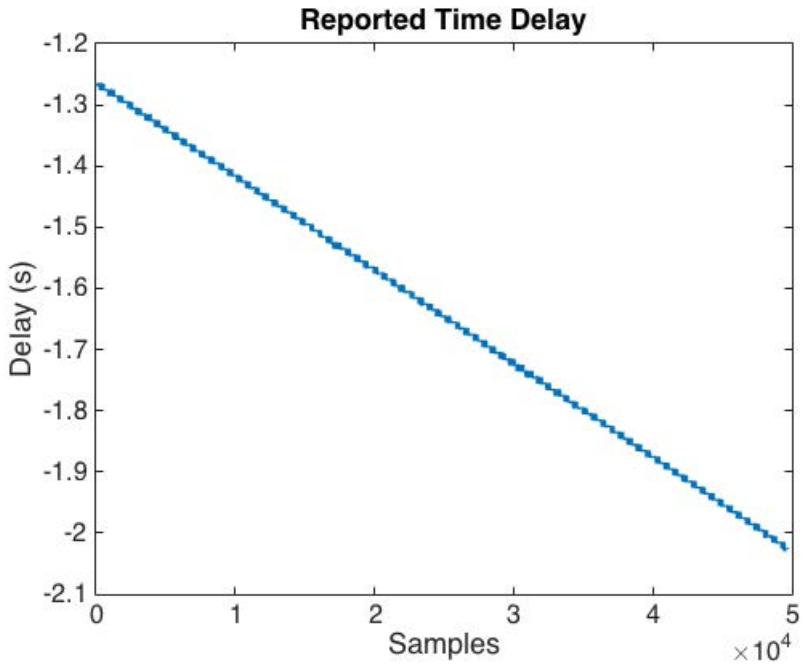

Figure 4.60: IMU Hardware Module Time Delay.

4.4.4 WSS Hardware Module

Testing of the WSS hardware module requires a process similar to that of the WSS software module. First, the wheel speed error for a 10 mph run is examined in Figures 4.61 and 4.62. As seen, the WSS hardware module yields error characteristics similar to both the WSS software module and the hardware WSS shown in Sections 4.2.3 and 4.3.4. Clock jitter again causes the maximum error to be more than one-half a quantization step, while the standard deviation of wheel counts exhibits linear growth with increasing speed. Figure 4.63 demonstrates the close match of standard deviation growth between the WSS hardware module and the hardware WSS. The slope of the fit for the WSS hardware module matches the slope of the hardware WSS fit perfectly. This is due to the fact the WSS hardware module is sampled in the exact same way as the hardware WSS.

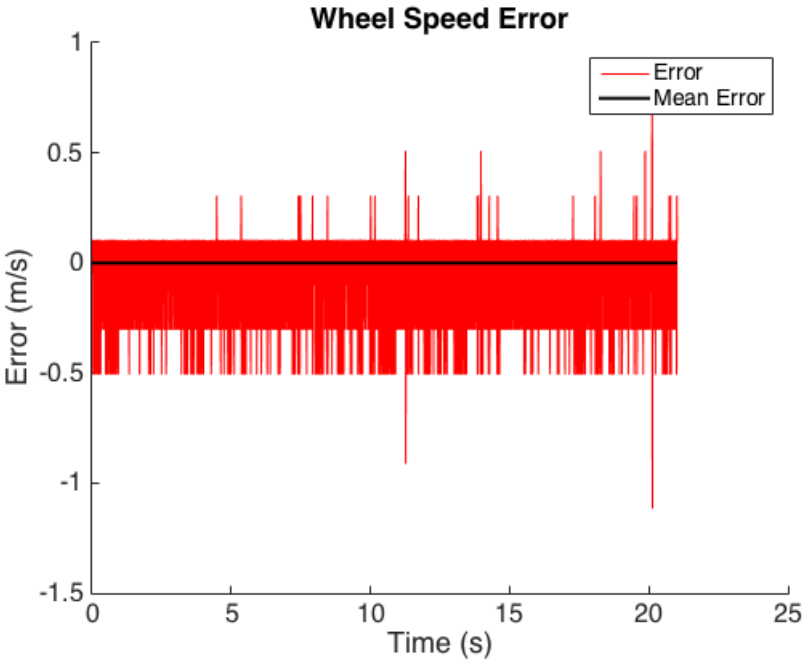

Figure 4.61: Wheel Speed Hardware Module Error at 10 mph.

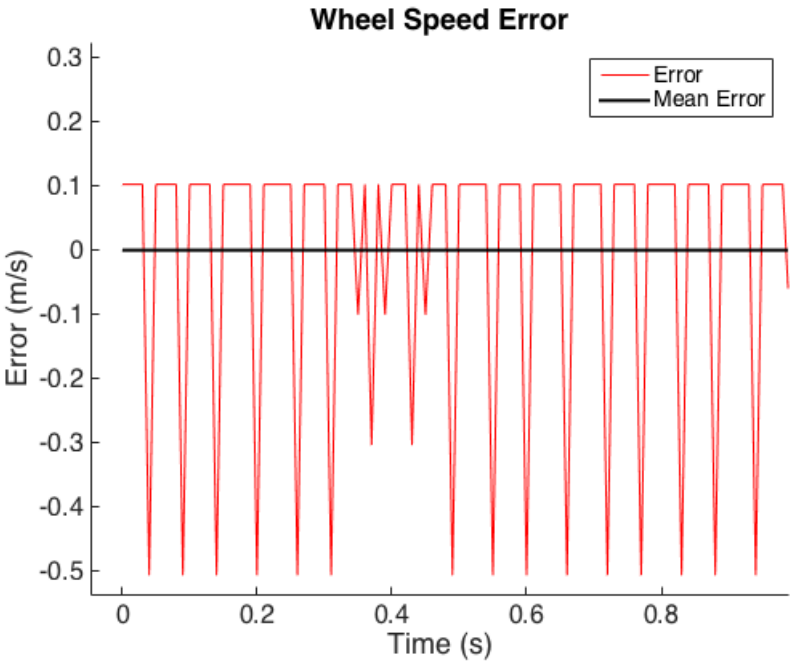

Figure 4.62: Close View of Wheel Speed Hardware Module Error at 10 mph.

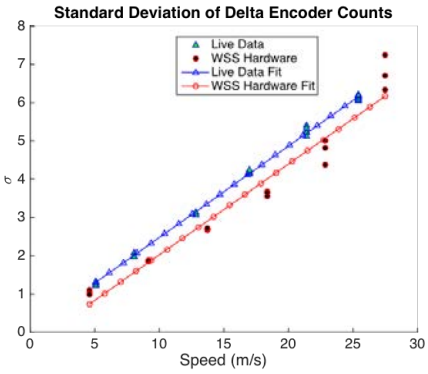

Figure 4.63: WSS Hardware Module Standard Deviation of Encoder Counts.

Examination of wheel speed error with increasing speed shows the same trend for the WSS hardware module as the software module. As speed increases, the wheel speeds report a higher velocity than the vehicle in actually traveling due to wheel slip. However, this error trend grows in the opposite direction of the experimental data, shown in Figure 4.64. This difference in error trends is similar to that observed in Section 4.3.4 for the WSS software module. Changing the wheel radius used to calculate experimental wheel speeds by just 2.8% like with the WSS software module produces the trends shown in Figure 4.65.

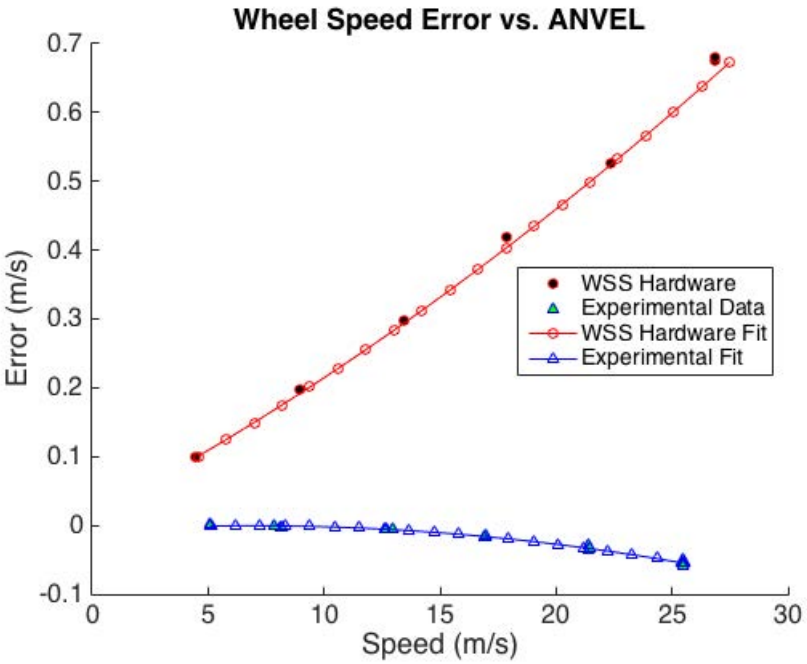

Figure 4.64: WSS Hardware Module Wheel Speed Error.

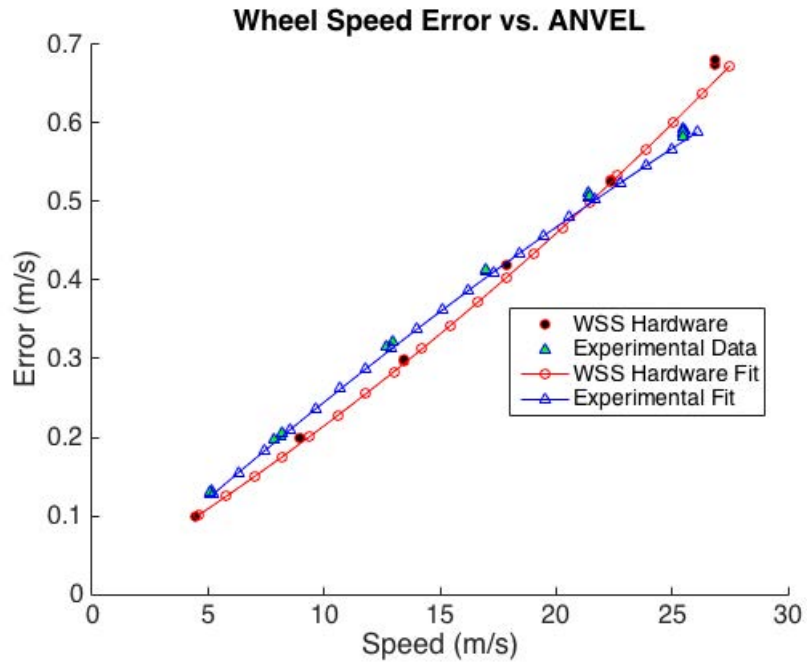

Figure 4.65: WSS Hardware Module Wheel Speed Error with Modified Radius.

The two error trends now match much more closely than when using the estimated wheel radius to calculate speed. Differences in slope between the two trends can still be observed, suggesting the difference is likely due to more than an incorrectly estimated wheel radius. One potential cause could be the radius of the wheel in the experimental data may be changing as speed increases but this effect is not modeled by ANVEL.

4.5 Conclusions

This chapter introduced the experimental setups used to test the developed models and characterize the GPS, IMU, and WSS used during live data collection. The experimental test characteristics found were then used as benchmarks for comparison of the errors produced in the developed modules. This chapter also demonstrated that all of the developed modules produce outputs that are compatible with the respective sensors being simulated. The software modules output data in the same message formats as the actual sensors, while the hardware modules output signals that are sampled in the exact fashion as signals from the actual sensors. Difficulties encountered during testing and validation were also documented, including time stamping issues and problems encountered employing the Spectracom simulator in the GPS hardware module.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis has presented the development of a Hardware In the Loop/Software In the Loop simulation environment with the goal of testing vehicle navigation units and algorithms for use in autonomous vehicles. The system uses ANVEL for vehicle simulation and ideal sensor measurements which may be used in software or hardware error model implementations for flexibility in testing. An introduction to various technologies used in autonomous vehicle navigation was presented in Chapter 2. The basics of GPS, inertial sensors, and GPS/INS navigation were discussed to provide a background for system development. Measurement principles and common error sources for each sensor type were introduced.

Chapter 3 presented the development of the simulation environment with considerations made to the topics discussed in Chapter 2. The overall system architecture and the interaction between ANVEL and the developed modules using plugins was introduced. A GPS software simulation was then developed as a plugin to ANVEL using position and velocity from ANVEL as the basis for measurement generation. The GPS simulation employed ephemeris and atmospheric data to make simulated outputs as realistic as possible. Simulated outputs include pseudorange, Doppler shift, carrier phase, and a PVT solution with characteristics that emulate a real GPS receiver. The major advantage of the GPS simulation is the ability to perform ray tracing and remove satellites blocked by environmental obstacles. Simple error models for accelerometers and gyroscopes were then introduced to approximate the behavior of the IMU used in the GAVLab's navigation unit using inertial data from ANVEL as true measurements.

The models are general such that they can be modified to emulate other IMUs as well. Finally, a model to output encoder counts with quantization error based on perfect wheel speeds from ANVEL was presented. Each sensor's software implementation was then presented in which the developed models output standard and custom ROS messages representative of the actual sensors being simulated. The software modules were then used as the bases of development for each sensor's respective hardware module in which outputs were sampled using the same code as the actual sensors. The GPS hardware module employed a Spectracom GPS simulator to output RF signal representative of the position and velocity reported by ANVEL which was then sampled using a Novatel receiver. The GPS hardware module also has the ability to block satellites based on satellite visibility information output by the GPS software module. The IMU hardware module used error generation from the software module and outputs data as a serial signal to emulate actual IMU output and is sampled using a serial to USB adapter and open source C++ code. The WSS hardware module received wheel speed data from ANVEL and used an Arduino Due's PWMs to generate quadrature wave forms that are sampled using a US Digital QSB adapter.

Chapter 4 detailed the testing and validation of the developed system. Characterization of the modeled sensors was performed using both static and dynamic data sets. The RMS errors of various Novatel measurements were presented as benchmarks for simulation comparison. Parameter identification methods such as the Allan variance and autocorrelation were discussed and employed as a means of extracting IMU error model parameters from experimental data. Magnitudes of quantization error and the error growth with speed for the wheel speed sensors was then presented. The identified parameters were then used as a basis of comparison for the sensor models to ensure the developed modules approximate real sensor behavior as accurately as possible. The performance of the GPS software module was first verified using a static data set simulating live-sky data recorded using a Novatel receiver and then using a dynamic set with a vehicle driving through a 3-D model of downtown Auburn, Alabama. IMU software and hardware modules were then verified by comparing Allan variances of experimental data and simulated data. Additionally, error characteristics found differencing simulated IMU data and perfect inertial measurements from ANVEL verified the performance of the IMU models.

Finally, the wheel speed modules were verified using simulations of the vehicle traveling at various speeds to mimic the characterization method used for the hardware WSS.

Several issues in the current implementation were also identified. Potential problems with time stamping in the IMU and WSS modules were identified possibly due to different time stamping methods being used between the developed modules and the ROS gateway. The iterative method necessary to accurately approximate the IMU with simple sensor models also presented problems, requiring many hours of data collection and analysis to produce acceptable model characteristics. Difficulties with Spectracom implementation were also discussed including a time bias between position command and output. Other difficulties with the Spectracom include a potential change in timing bias during dynamic testing and loss of tracking during movement. However, the overall system should be capable of providing researchers a HIL/SIL test bed to use in navigation unit development with confidence their algorithms will perform similarly in experimental application.

The major benefit of the system architecture presented in this work is the modularity. Researchers may choose to use only the software modules for rapid prototyping without the added complexity of the hardware modules. Alternatively, users may choose to use the hardware modules to add more realistic errors inherent to hardware sensor implementation. Users can test with actual hardware in the simulation loop or test navigation algorithms with SIL. Any combination of software and hardware modules may be used with simulated errors on or off, giving researchers the ability to easily investigate the effects of individual error sources. The modular architecture also lends itself well to the addition of other sensor packages such as LiDAR or cameras. Using models verified by experimental data also gives researchers more confidence their algorithm will behave similarly during hardware implementation and allows the researcher to use the HIL/SIL system further in the development process, thus reducing risks inherent to hardware testing. Employing real ephemeris and atmospheric data along with ray tracing in GPS simulations allows users to test algorithms in GPS degraded environments, typically difficult to do in simulation. Finally, the system may be easily adapted for use with any navigation unit or algorithm, provided the algorithm uses the same ROS message types output by the developed modules.

5.2 Future Work

5.2.1 Current System

This work has identified several opportunities for improvement. First, issues with the current implementation should be addressed. To solve timing issues, a time server could be setup to ensure that the ANVEL PC and the Linux PC operate using the same clock. Network Time Protocol (NTP) may be a good candidate for time synchronization due to the fact NTP is built in to Windows and most Linux implementations. Additionally, changing the time stamping method within the ROS gateway provided by TARDEC could further reduce timing issues.

An investigation into timing issues within the GPS hardware module should also be performed. It may be possible to reduce the time bias shown in Section 3.6. Also, potential changes in the timing bias observed during dynamic testing using the Spectracom in Section 4.4.1 should be investigated to ensure the Spectracom produces useful signals during dynamic testing. Furthermore, the large errors in pseudorange and Doppler shift shown during Spectracom integration should be investigated further. Live-sky data could be recorded with a receiver capable of outputting a clock solution and the same data set then simulated using the Spectracom. Clock solutions for the different runs would then be compared, likely explaining differences in pseudorange and Doppler measurements.

Another area of improvement could be improving the fidelity of the error models for each module. The error models presented in this work are meant to be used as a base for further development and were shown to provide a good first cut approximation of the sensors being simulated, but did not fully capture all of the error dynamics of each sensor. Examples of error model improvements include accounting for bias instability in the IMU models and incorporation of a receiver clock or multipath error model in the GPS software module. Validation with longer data sets to more accurately capture error dynamics with large time constants for each sensor set is another area for improvement.

5.2.2 Further Development

As previously stated, the system detailed in this thesis was developed using an open framework with future development in mind. Extensions to the current implementation would increase system capabilities. One useful extension for the GPS software module would be the incorporation of signals from other satellite constellations such as GLONASS and GALILEO. Other extensions include Signals of OPportunity (SOP) to increase available ranging signals and the implementation of a simulated base station to provide the simulated receiver with RTK corrections.

Other sensor types commonly used for navigation could also be added to the HIL/SIL system, such as LiDAR, radar, and cameras. Addition of these sensors would allow researchers to test Simultaneous Localization and Mapping (SLAM) algorithms integrated with the GPS/IMU/WSS navigation system from this thesis in simulation prior to hardware implementation. Also, only the sensors used in the GAVLab navigation unit specifically were modeled and implemented. The addition of error models for other sensors will expand the capabilities of the system and would allow for testing with a variety of sensor grades to examine how algorithm performance changes. Adding other sensors to the system, especially in HIL implementation, allows for testing of navigation units from various vendors. A simulation could be run using one vendor's navigation unit immediately following a simulation using another vendor's navigation unit. This would allow for a direct comparison of navigation units and algorithms in a laboratory environment

References

[1] D. W. Allan. Statistics of atomic frequency standards. *Proceedings of the IEEE*, 54(2):221–230, Feb 1966.

[2] GGOS Atmosphere. Vmfg tropospheric grid. Available on-line, 2018.

[3] Andreas Bayha, Franziska Grüneis, and Bernhard Schätz. Model-based software in-the-loop-test of autonomous systems. In *Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, 2012.

[4] John W. Betts. *Engineering Satellite-Based Navigation and Timing*. Wiley, oboken, New Jersey, 2016.

[5] H. Black and A. Eisner. Correcting satellite doppler data for tropospheric effects. *Journal of Geophysical Research*, 89:2616–2626, 1984.

[6] Johannes Böhm, Birgit Werl, and Harald Schuh. Troposphere mapping functions for gps and vlbi from ecmwf operational analysis data. *Journal of Geophysical Research*, 111, 2006.

[7] Lowell Brown and D. Edwards. *GNSS for Vehicle Control*, chapter Vehicle Modeling, pages 61–89. Artech House, Norwood, MA, 2010.

[8] J. Blake Bullock. *Understanding GPS Principles and Applications*, chapter Integration of GPS with Other Sensors and Network Assistance, pages 459–558. Artech House, 2nd edition, 2006.

[9] Christopher R. Carlson, J. Christian Gerdes, and J. David Powell. Error sources when land vehicle dead reckoning with differential wheelspeeds. *NAVIGATION, Journal of the Institute of Navigation*, 51(1):13–28, 2004.

[10] John Paul Collins. Assessment and development of a tropospheric delay model for aircraft users of the global positioning system. Master's thesis, University of New Brunswick, New Brunswick, Canada, September 1999.

[11] Crustal Dynamics Data Information System (CDDIS DAAC). Broadcast ephemeris data. Available on-line, 2018.

[12] Crustal Dynamics Data Information System (CDDIS DAAC). Ionosphere total electron content maps. Available on-line, 2018.

[13] A. Dempster. Correlators for l2c: Some considerations. *Inside GNSS*, 1(7):32–37, October 2006.

[14] Lei Dong. If gps simulator development and verification. Master's thesis, University of Calgary, Calgary, Alberta, Canada, December 2003.

[15] MaryAnne Fields, Ralph Brewer, Harris L. Edge, Jason L. Pusey, Ed Weller, Dilip G. Patel, and Charles A. DiBerardino. Simulation tools for robotics research and assessment. *Proc.SPIE*, 2016.

[16] Warren S. Flenniken. Modeling inertial measurement units and analyzing the effect of their error in navigation applications. Master's thesis, Auburn University, Auburn, Alabama, December 2005.

[17] Karl Flueckiger, Don Gustafson, and John Dowdle. Ins/gps deep integration navigation hardware testbed. In *Proceedings of the 61st Annual Meeting of The Institute of Navigation (2005*, pages 1171–1178, Cambridge, MA, June 2005.

[18] J. R. Fountain. Characteristics and overview of a silicon vibrating structure gyroscope. In *NATO RTO Lecture Series 232 "Advances in Navigation Sensors and Integration Technology", 31 May - 1*, London, U.K., October 2003.

[19] R. L. French. *Global Positioning System: Theory and Applications Volume II*, chapter Land Vehicle Navigation and Tracking, pages 275–301. AIAA, Washington, D.C., 1996.

[20] C. Galko, R. Rossi, and X. Savatier. Vehicle-hardware-in-the-loop system for adas prototyping and validation. In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, pages 329–334, July 2014.

[21] D. Gebre-Egziabher. *Design and performance analysis of a low-cost aided dead-reckoning navigation*. PhD thesis, Stanford, 2004.

[22] A. Gelb. *Applied Optimal Estimation*. M.I.T. Press, 1974.

[23] Olaf Gietelink, Jeroen Ploeg, Bart De Schutter, and Michel Verhaegen. Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. *Vehicle System Dynamics*, 44(7):569–590, 2006.

[24] M. Grewal, L. Weill, and A. Andrews. *Global Positioning Systems, Inertial Navigation, and Integration*. John Wiley and Sons, Inc., 2001.

[25] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. Artech House, Boston, Massachusetts, 2013.

[26] John A. Gubner. *Probability and Random Processes for Electrical and Computer Engineers*, pages 150–151. Cambridge University Press, 2006.

[27] J. Hao and S. Guo. The study of dual frequency ionospheric error correction method and accuracy analysis based on gps. In *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4, Sept 2011.

[28] C. Hay. Turn, turn, turn: Wheel-speed dead reckoning for vehicle navigation. *GPS World*, pages 37–42, October 2005.

[29] Sebastian Hening, Corey Ippolito, Kalmanje S. Krishnakumar, Vahram Stepanyan, and Mircea Teodorescu. 3d lidar slam integration with gps/ins for uavs in urban gps-degraded

environments. In *AIAA Information Systems-AIAA Infotech @ Aerospace, AIAA SciTech Forum*, 2017.

[30] Ch. Hollenstein, E. Favey, C. Schmid, A. Somieski, and D. Ammann. Performance of a low-cost real-time navigation system using single-frequency gnss measurements combined with wheel-tick data. In *ION GNSS 2008*, pages 1610 – 1618, Savannah, GA, September 2008.

[31] Christopher R. Hudson, Alexander Lalejini, Brandon Odom, Cindy L. Bethel, Daniel W. Carruth, Phillip J. Durst, and Christopher Goodin. Anvel-ros: The integration of the robot operating system with a high-fidelity simulator. In *2015 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM*, 2015.

[32] J.M. Juan Zornoza J. Sanz Subirana and M. Hernández-Pajares. Klobuchar ionospheric model, 2011.

[33] D. Karnick, G. Ballas, L. Koland, M. Secord, T. Braman, and T. Kourepenis. Honeywell gun-hard inertial measurement unit (imu) development. In *PLANS 2004. Position Location and Navigation Symposium (IEEE Cat. No.04CH37556)*, pages 49–55, April 2004.

[34] V. Kempe. *Inertial MEMS Principles and Practices*. Cambridge University Press, Cambridge, U.K., 2011.

[35] P. J. King and D. G. Copp. Hardware in the loop for automotive vehicle control systems development. In *2004 Mini Symposia UKACC Control*, pages 75–78, Sept 2004.

[36] John A. Klobuchar. Ionospheric time-delay algorithm for single-frequency gps users. *IEEE Transactions on Aerospace and Electronic Systems*, AES-23(3):325–331, May 1987.

[37] P. Lepej, A. Santamaria-Navarro, and J. Solà. A flexible hardware-in-the-loop architecture for uavs. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1751–1756, June 2017.

[38] Scott Martin. Closely couplde gps/ins relative positioning for automated vehicle convoys. Master's thesis, Auburn University, Auburn, Alabama, 2011.

[39] Pratap Misra and Per Enge. *Global Positioning System - Signals, Measurements, and Performance*. Ganga-Jamuna Press, Lincoln, Massachesetts, 2011.

[40] Navstaer Global Positioning System. *Interface Specification IS-GPS-200-D*, revision d edition, March 2006.

[41] Brently Nelson, David Bevly, Jeff Ratowski, and Bernard Theisen. Anvel hil/sil additions for navigation algorithm development. In *2017 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM*. NDIA Michigan, 2017.

[42] M. Odelga, P. Stegagno, H. H. Bülthoff, and A. Ahmad. A setup for multi-uav hardware-in-the-loop simulations. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 204–210, Nov 2015.

[43] Russell Powell. A multiple-antenna software gps signal simulator for rapid testing of interference mitigation techniques. Master's thesis, Auburn University, Auburn University, December 2016.

[44] LLC Quantum Signal.

[45] J. Rankin. An error model for sensor simulation gps and differential gps. In *Position Location and Navigation Symposium, 1994., IEEE*, pages 260–266, Apr 1994.

[46] RTCA-MOPS. *GLOBAL POSITIONING SYSTEM WIDE AREA AUGMENTATION SYSTEM (WAAS) PERFORMANCE STANDARD*, 2006.

[47] M. Kamel Salaani, David Mikesell, Chris Boday, and Devin Elsasser. Heavy vehicle hardware-in-the-loop automatic emergency braking simulation with experimental validation. *SAE International Journal of Commercial Vehicles*, 9(2):57–62, 2016.

[48] Daniel Cody Salmon. An experimental exploration of low-cost sensor and vehicle model solutions for precision ground vehicle navigation. Master's thesis, Auburn University, Auburn, Alabama, December 2015.

[49] Kristin E Schaefer, Ralph W Brewer, E Ray Pursel, Anthony Zimmermann, and Eduardo Cerame. Advancements made to the wingman software-in-the-loop (sil) simulation: How to operate the sil. Technical report, US Army Research Laboratory, December 2017.

[50] Kristin E Schaefer, Ralph W Brewer, E Ray Pursel, Anthony Zimmermann, Eduardo Cerame, and Keith Briggs. Outcomes from the first wingman softwarein-the-loop integration event: January 2017. Technical report, US Army Research Laboratory, June 2017.

[51] G. Seeber. *Satellite Geodesy*. Walter De Gruyter Inc, 2003.

[52] Spectracom. Gsg - 5/6 advanced gnss simulators. Web.

[53] Stephen Steffes, Malak Samaan, Michael Conradt, and Stephan Theil. *Reconfigurable Hardware-in-the-Loop Test Bench for the SHEFEX2 Hybrid Navigation System Experiment*. American Institute of Aeronautics and Astronautics, 2018/04/17 2011.

[54] J. Stephen and G. Lachapelle. Development and testing of a gps-augmented multi-sensor vehicle navigation system. *Journal of Navigation*, 54(2):297–319, 2001.

[55] J. Sanz Subirana, J.M. Juan Zornoza, and M. Hernández-Pajares. Tropospheric delay, 2011.

[56] K. S. Swanson, A. A. Brown, S. N. Brennan, and C. M. LaJambe. Extending driving simulator capabilities toward hardware-in-the-loop testbeds and remote vehicle interfaces. In *2013 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pages 115–120, June 2013.

[57] T.A.C. Verschuren. Extracting more accurate position and velocity estimations using time stamping. Master's thesis, Eindhoven University of Technology, June 2006.

[58] John Wall. A study of the effects of stochastic inertial sensor errors in dead-reckoning navigation. Master's thesis, Auburn University, August 2007.

[59] J. L. Wilson and M. J. Slade. Accelerometer compensated differential wheel pulse based dead reckoning. In *Proc. ION GNSS 2009*, pages 3087–3095, Savannah, GA, September 2009.

[60] Jefferey L. Wilson. Low-cost pnd dead reckoning using automotive diagnostic links. In *ION GNSS 2007*, pages 2066 – 2074, Fort Worth, TX, September 2007.

[61] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. Building a ros-based testbed for realistic multi-robot simulation: Taking the exploration as an example. *Robotics*, 6, 2017.

[62] Y. Zhao. *Vehicle Location and Navigation Systems*. Artech House, Norwood, MA, 1997.

Appendices

Appendix A

Satellite Position Calculation

First, the satellites mean motion n_k is calculated using Equation (A.1)

$$n_k = \sqrt{\frac{\mu}{a_k^2}} + \Delta n_k \tag{A.1}$$

where μ is the Earth's gravitational constant (μ = 3.986005 $\times 10^{14}$ m^3/s^2), a_k is the satellite's semi-major axis (a_k = $(\sqrt{a_k})^2$), and Δn_k is the mean motion difference. Next, the transmit time t_{t_k} is corrected to account for the difference between the ephemeris epoch time t_{oe_k} and possible GPS week crossovers in Equation (A.2)

$$t_{t_k} = \begin{cases} t_{t_k} - 604.800, & \text{if } t_{t_k} - t_{oe_k} > \frac{604{,}800}{2} \\ t_{t_k} + 604.800, & \text{if } t_{t_k} - t_{oe_k} < \frac{-604{,}800}{2} \end{cases} \tag{A.2}$$

The mean anomaly M_k is then found using Equation (A.3)

$$M_k = M_{0_k} + n(t_{t_k} - t_{oe_k}) \tag{A.3}$$

where M_{0_k} is the mean anomaly at the ephemeris reference time. Next, the non-linear eccentric anomaly E is calculated using an iterative approach with the initial guess E_{k_i} set the the previously calculated mean anomaly M_k. Calculation of the next eccentric anomaly value is performed using Equation (A.4)

$$E_{k_i+1} = M_k + e_k \sin E_{k_i} \tag{A.4}$$

where e_k is the eccentricity provided in the satellite ephemeris. If the difference between E_{k+1} and E_k is less than a predetermined value, E_{k+1} is set as the eccentric anomaly, otherwise the process is repeated until the difference is small enough. Next, the satellite clock correction δt_{s_k} is calculated using Equation (A.5)

$$\delta t_{s_k} = a_{f0_k} + a_{f1_k}(t_{t_k} - t_{oc_k}) + a_{f2_k}(t_{t_k} - t_{oc_k}) + Fe_k\sqrt{a_k}\sin E_k - T_{GD_k} \tag{A.5}$$

where a_{f0_k}, a_{f1_k}, and a_{f2_k} are the satellite clock corrections, t_{oc_k} is the time of ephemeris, and T_{GD_k} is the total group delay, all broadcast in the ephemeris. The term $Fe_k\sqrt{a_k}\sin E_k$ with F = -4.442807633 $\times 10^{-10}$ $s/m^{1/2}$ from Equation (A.5) accounts for the relativistic effects between the satellite and user. The calculated satellite clock correction is used to incorporate satellite clock error into the simulated pseudoranges, discussed in section 3.3. The signal transmit time is now recalculated to incorporate the satellite clock correction in Equation (A.6).

$$t_{t_k} = t_{t_k} - \delta t_{s_k} \tag{A.6}$$

The true anomaly ν_k is calculated using Equation (A.7)

$$\nu_k = \arctan \frac{\sqrt{1-e_k^2}\sin E_k/(1 - e_k\cos E_k)}{(\cos E_k - e_k)/(1 - e_k\cos e_k)} \tag{A.7}$$

and the argument of latitude ϕ_k is calculated using Equation (A.15)

$$\phi_k = \nu_k + \omega_k \tag{A.8}$$

where ω_k is the argument of perigee from the satellite ephemeris. Corrections to the argument of latitude, radius r_k, and inclination i_k are then calculated using Equations eqs. (A.9) to (A.11)

$$\delta\phi_k = C_{us_k}\sin 2\phi_k + C_{uc_k}\cos 2\phi_k \tag{A.9}$$

$$\delta r_k = C_{rs_k}\sin 2\phi_k + C_{rc_k}\cos 2\phi_k \tag{A.10}$$

$$\delta i_k = C_{is_k}\sin 2\phi_k + C_{ic_k}\cos 2\phi_k \tag{A.11}$$

and used to correct in Equations eqs. (A.12) to (A.14)

$$\phi_k = \phi_k + \delta\phi_k \tag{A.12}$$

$$r_k = a + k(1 - e_k \cos E_k) + \delta r_k \tag{A.13}$$

$$i_k = i_k + \delta i_k + idot_k(t_{t_k} - t_{oe_k}) \tag{A.14}$$

where a_k represents the semi-major axis and $idot$ is the inclination angle rate provided in the ephemeris. Next, the longitude of the ascending node Ω_k is corrected to account for the angle between the ascending node and the Greenwich Meridian using (A.15)

$$\Omega_k = \Omega_k + t_{t_k}(\dot{\Omega}_k - w_e) - w_e t_{oe_k} \tag{A.15}$$

Finally, the position for satellite k, $\bar{r}_{s_k}$, may be calculated using Equation (A.16)

$$\bar{r}_{s_k} = \begin{bmatrix} x_{s_k} \\ y_{s_k} \\ z_{s_k} \end{bmatrix} = \begin{bmatrix} r_k \cos\Omega_k \cos\phi_k - r \sin\Omega_k \cos i_k \sin\phi_k \\ r_k \sin\Omega_k \cos\phi_k + r \cos\Omega_k \cos i_k \sin\phi_k \\ r_k \sin i_k \sin\phi_k \end{bmatrix} \tag{A.16}$$

The satellite positions $\bar{r}_{s_k}$ and the clock correction term δt_{s_k} are generated at each epoch of the scenario simulation and are used to model the user-satellite pseudoranges detailed in Section 3.3.