Covariance Analysis for Multi-Source Navigation Architecture

THESIS

Tristan T. Williams

AFIT-ENG-MS-22-M-073

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-22-M-073

Covariance Analysis for Multi-Source Navigation Architecture

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Tristan T. Williams, B.S.E.E.

March 19, 2022

AFIT-ENG-MS-22-M-073

Covariance Analysis for Multi-Source Navigation Architecture

THESIS

Tristan T. Williams, B.S.E.E.

Committee Membership:

Robert C. Leishman, Ph.D
Chair

Maj. J. Curro, Ph.D
Member

Randy Christiensen, Ph.D
Member

AFIT-ENG-MS-22-M-073

Abstract

Currently, analysis on navigation systems can be slow and computationally expensive using Monte Carlo approaches. Covariance analysis is a tool that can return trade-space analysis results promptly and can be computationally cheap. Covariance analysis tools are mostly ad-hoc or within proprietary interfaces. This is especially true for the realm of navigation, as most covariance analysis papers deal with a single scenario and write an ad-hoc simulator for said scenario [1, 2].

This research aims to create a covariance analysis tool in a new modular and pluggable navigation framework library, the Navigation Toolkit. The Navigation Toolkit is a government-reference library that can be used out of the box with the modular and pluggable sensor fusion architecture known as pntOS. Creating a covariance analysis tool inside a modular and pluggable navigation software package will allow researchers to quickly obtain trade-space analysis results and easily conduct their own covariance analysis simulations using largely the same code that is used for real sensor fusion. Researchers will be able to leverage a large amount of sensor models, algorithms, and filters that come prepackaged within Navigation Toolkit.

The creation of this covariance tool is explained through the analysis of two different navigation scenarios. Results from these different navigation scenarios are explored to determine the benefits and drawbacks behind the sensor combinations. A new Doppler LiDAR velocity sensor is first evaluated with a short, four hundred second set of simulated flight data to prove the functionality of the covariance tool and to simultaneously show the capabilities of the new sensor. The final evaluation is conducted using 3 hours of flight data. This scenario pairs the Doppler LiDAR velocity sensor with a high-sensitivity scalar magnetometer to understand how the combination of these two sensors may improve a navigation solution.

Acknowledgements

I would like to thank my advisor, Dr. Leishman for his guidance throughout this process and encouragement along the way. I would also like to thank my committee for their insight and time. Finally, I would like to thank my beautiful girlfriend for all of her support and patience throughout this process.

Tristan T. Williams

Table of Contents

Page

List of Figures

List of Tables

Covariance Analysis for Multi-Source Navigation Architecture

I. Introduction

1.1 Problem Background

1.1.1 Trade-Space Analysis

The most commonly used method of trade-space analysis is the Monte Carlo method [6]. The Monte Carlo method performs a large number of simulation runs of a system while perturbing parameters to obtain samples to populate a distribution of results. The average value is understood to be the most likely outcome of the simulation. Monte Carlo analysis can require many hundreds or thousands of runs to achieve accurate results depending on the complexity of the problem at hand. These runs can take days to complete, depending on the complexity of the problem being simulated and the number of parameters being varied. Covariance analysis can, in certain circumstances and under some assumptions, provide the same statistical results as a Monte Carlo simulation without large amounts of computation time and in as few as one simulation run [7, 6]. Covariance analysis provides an estimate of system uncertainties with time, given the user has the required truth model of the system. This thesis documents the creation of an analysis tool that will aid researchers in the quick creation and analysis of covariance analysis simulations. Then, describes the use of the tool with two different use cases.

1.1.2 Position, Navigation, and Timing Operating System (pntOS)

pntOS is a pluggable Bayesian estimation application that allows a software framework to generate navigation simulations and scenarios in a virtual environment. A library exists alongside pntOS, called Navigation Toolkit, that allows for users to utilize modular Bayesian estimators and assemble plugins that are compatible with pntOS. The overall architecture creates an environment where users can create different filters, sensors, and algorithms that can be easily accessed and used in a wide variety of applications.

1.2 Project Motivation

1.2.1 Covariance Analysis Tool

The creation of a covariance analysis tool inside of the Navigation Toolkit software package will allow researchers to conduct covariance analysis quickly and easily, largely with the same code that will be used for the actual navigation computations. With the speed of technology invention today, the creation of new ways to navigate Earth are ever-changing. One problem researchers and government offices have is the task of sorting through many potential sensors to determine how they might best utilize these new developments in technology. The proposed covariance analysis tool will directly aid in this challenge. With this tool, a researcher may not need to simulate a single scenario thousands of times to estimate the navigation state covariance. A single run will be required of the simulation to get the covariance bounds of the navigation state.

A tool that researchers can use to conduct navigation covariance analysis will speed up early stage research. Researchers given a new sensor could test in the field or use a simulation first. The fastest way would be to build an entire simulation by hand and run it thousands of times to create a best estimate of how the system would respond. This would allow the researcher to have a better understanding of how the sensor would contribute to a navigation solution and given different simulations, tell many different characteristics of the new sensor. With the covariance analysis tool, the researcher would simply have to plug in their sensor sigmas and direct the code to the location of the truth data they wish to use for the simulation. Then in a fraction of the time as a Monte Carlo simulation, the researcher will have the error state plots and can determine if their new sensor or new navigation setup is what they want it to be.

A covariance analysis can be run with a simulation flight across multiple trajectories as well as different inertial measurement unit (IMU) grades to understand the sensor's contribution to a navigation system. With a full simulated analysis of a sensor, it will be much easier for researchers and others to understand the full benefits that a particular sensor might bring to a navigation solution. We will be conducting a covariance analysis to aid in our understanding of new sensors and will give a brief explanation of what covariance analysis is and how it can be used to obtain results for new sensors, such as, the Psionic Doppler Light Detection And Ranging (LiDAR) sensor.

1.2.2 Case Studies

This paper will conduct covariance analysis on two different sensors to demonstrate how the proposed tool can help researchers conduct quick and efficient trade-space analyses. These case studies will help describe how covariance analysis works and will be beneficial for the emerging sensor technologies.

A novel Doppler LiDAR sensor was originally developed by NASA and has been transferred to a company, Psionic, for further development and maturation. This unique sensor can make highly accurate, three-dimensional velocity measurements in the body frame of a vehicle. The company states the sensor can produce measurements with covariance around 3 cm/sec at a rate in the kilohertz.

Magnetic anomaly navigation is a Global Positioning System (GPS)-alternative navigation method that works by matching magnetometer measurements from a vehicle's magnetometer sensor onto a map of the Earth's magnetic anomaly field [5]. Magnetic Anomaly Navigation (MagNav) is a form of navigation is a nearly unjammable passive navigation system that works in any environment, weather conditions, or time of day. MagNav is a method that has shown many recent promising results in how it can aid a GPS-denied navigation scenario [5, 8, 9, 10]. MagNav could benefit from more published simulations displaying the advantages, and we can provide this through the use of our covariance analysis tool.

1.3 Document Overview

This work begins in Chapter II with the background of the software suite used to create the covariance analysis tool in Chapter II. Chapter II will also describe the use cases we will analyze in future sections using the covariance analysis tool. The following chapters, Chapter III and Chapter IV, are individual papers that present the two different use cases of the covariance analysis tool. Chapter Chapter III was presented at National Aerospace and Electronics Conference (NAECON) in 2021 [11]. The next chapter, Chapter IV incorporates the Doppler LIDAR velocity sensor from before with a MagNav sensor to demonstrate the solution the two sensors can create in a combined navigation solution and will be submitted to the Journal of the Institute of Navigation. Finally, Chapter V summarizes the findings and the creation of the tool, then finishes with possible future work.

II. Background and Literature Review

This chapter presents the fundamental background information used to support the software design decisions in the subsequent chapters. It is also utilized to create an understanding of covariance analysis and other forms of trade space analysis. Section 2.1 describes Kalman filtering and Section 2.2 explains extended Kalman filtering. Section 2.3 explains covariance analysis and Monte Carlo analysis. Section 2.4 describes pntOS, the navigation architecture that the covariance analysis tool is built within. This section also describes the models inside of pntOS that are utilized for covariance analysis. Section 2.5 describes the covariance analysis tool and how it was implemented into the pntOS architecture. Finally, Section 2.6 discusses related research conducted by various other authors and how this paper contributes to the field.

2.1 Kalman Filtering

An understanding of Kalman filtering is required to understand how we will be estimating a simulated flight's location with sensors. Kalman filtering is an algorithm that uses a series of measurements observed over time with noise and creates an estimate of the unknown variables. In navigation, we use the Kalman filter with sensors providing our measurements and states like position, velocity, and attitude being our unknown variables. The covariance analysis filter will be utilizing an extended Kalman filter that will be desribed in the following section. The following example will deal with a basic one-dimensional Kalman filter. A Kalman filter can be split into 2 parts, the state update equations, and the dynamic model equations [12, 13]. We will start by defining the equation to create the Kalman gain, as this is required to write the state update equation.

$$K_n = \frac{\text{Uncertainty in the Estimate}}{\text{Uncertainty in the Estimate} + \text{Uncertainty in the Measurement}} = \frac{p_{t,t-1}}{p_{t,t-1} + r_t} \tag{1}$$

where $p_{t,t-1}$ is the extrapolated estimate uncertainty that was calculated during the previous filter estimation and r_t is the measurement uncertainty at time t. The Kalman gain is defined as a number between 0 and 1, and with this we can write the state update equation as

$$\hat{x}_{t,t} = \hat{x}_{t,t-1} + K_t(z_t - \hat{x}_{t,t-1}) = (1 - K_t)\hat{x}_{t,t-1} + K_t z_t, \tag{2}$$

where $\hat{x}_{t,t}$ is the state estimate updated at time-step t. The Kalman gain K_t is the weight or accuracy assigned to the measurement. Inversely, $(1 - K_t)$ is the weight or accuracy given to the estimate of the state. The Kalman gain effects how much the predicted estimate $\hat{x}_{t,t-1}$ changes given the measurement just received z_t. Next, the definition of the estimate uncertainty update is

$$\hat{p}_{t,t} = (1 - K_t)\hat{p}_{t,t-1}, \tag{3}$$

which is the covariance update equation. As accuracy in the measurements increases, the uncertainty in the filters prediction decreases. Of course, with more and more iterations of the Kalman filter, we will increase the accuracy of the estimate until the filter achieves a steady-state balance between the prediction widening the state and the measurement updates shrinking them back down. The estimate uncertainty extrapolation is created with the dynamic model equations. These equations differ based on what one is estimating and can range from a large list of equations that are very complex to being constant. The number of dynamic model equations is directly related to the number of states that are being estimated. Figure 16 illustrates a diagram showing the flow of a Kalman filter. We will later use a Monte Carlo method as well as the covariance analysis to estimate how accurate the different sensor suites are at estimating the position and other states of a simulated aircraft. The Monte Carlo method utilizes the Kalman filter by running a large amount of the Kalman filters estimating the same scenario, with each run being an independent sample within the trajectory and sensors. Then, the individual Kalman filter runs are taken and statistics of those runs can be extracted to understand the anticipated result.

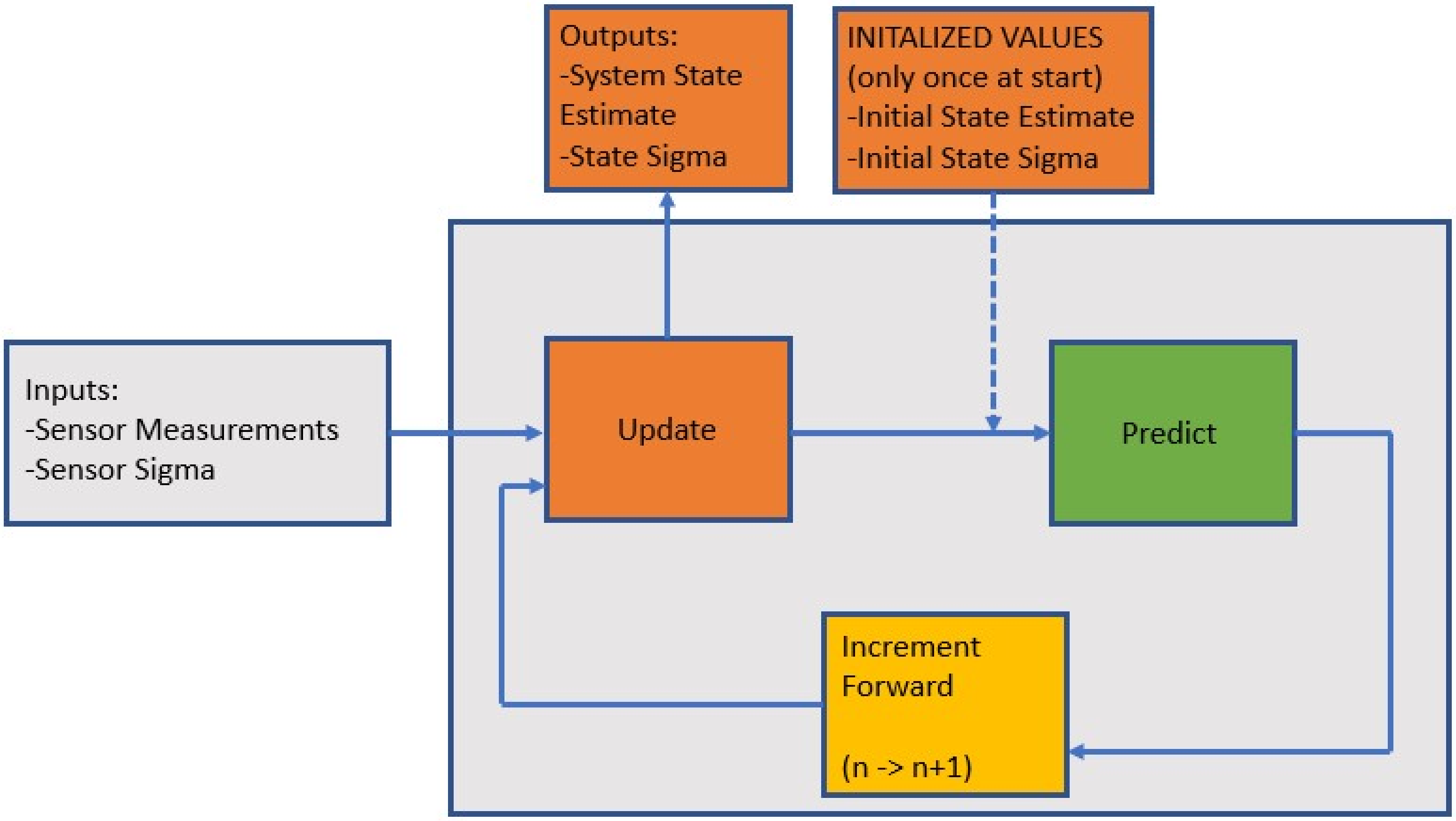


Figure 1: A visualization of a basic Kalman filter. Kalman filters are used to fuse mulitple measurements and estimate unknown variables. The fusion produces results that are more accurate than those based on a single measurement alone.

Now that the most basic understanding of the Kalman filter is developed, we can transition into an understanding of the Kalman filter within the realm of navigation.

The following will be a development of the Kalman filter as before but describing a basic navigation solution when using a Kalman filter. This development also assumes a state vector x, rather than a scalar

$$\boldsymbol{\dot{x}(t) = Fx(t) + Bu(t) + Gw(t),} \tag{4}$$

where $x(t)$ is the system state vector at a given time, $u(t)$ is the system input vector at a time, and $w(t)$ is the included white noise components vector at the same given time. The remaining variables F, B, and G are Jacobian matrices filled with constant coefficients. If we were using a basic discrete linear model, it would be given by

$$z_k = Hx_t + v_t, \tag{5}$$

with z being the given sensor measurement, H the observation model that is used to map the given measurements to states, and v is the white noise of the sensor. t is the the current timestamp of the system during the update of the state estimate. This is the most basic version of a discrete linear sensor when integrated into a Kalman filter. This model can be expanded to allow the fusion of more complex/non-linear sensors in a navigation solution. System states of the navigation solution are estimated by propagating the following

$$\hat{x}_{t+1}^{-} = \Phi\hat{x}_t^{+} + B_d u_t \tag{6}$$

$$P_{t+1}^{-} = \Phi P_t^{+} \Phi^T + Q_d \tag{7}$$

where x is the state estimate given by the Kalman filter, ϕ is the discrete state transition matrix, and P is the state error covariance matrix that is associated with the sensor. As Kalman filters work by propagating and updating, we can create our states and covariance by a combination of the estimates from the Kalman filter and the measurement readings from the sensors by introducing the Kalman gain below

$$K_k = P_t^- H^T [H P_t H^T + R]^{-1} \tag{8}$$

$$\hat{x}_t^+ = \hat{x}_t^- + K_k [z_t - H \hat{x}_t^-] \tag{9}$$

$$P_t^+ = (1 - K_t H) P_t^- \tag{10}$$

Finally, R is the sensor measurement error covariance matrix. This represents what happens in a single iteration of the Kalman filter and this continues over all time of the flight or simulation. Next, we will look at the Kalman Filter type that the covariance simulation will utilize. This is known as an Extended Kalman Filter or EKF.

2.2 Extended Kalman Filtering

Extended Kalman filters are Kalman filters that are extended to include non-linear models for state dynamics and sensor measurement models. These models are linearized to propagate covariance information, in the same way as the Kalman filter, but nonlinear functions are used to propagate mean values [14]. The system dynamics become the following:

$$\dot{x}(t) = f(x(t), u(t), t) + G(t) w(t). \tag{11}$$

where f is the nonlinear function that describes how the states change with time. The non-linear measurement model is

$$z_k = h(x_t, t_t) + v_t. \tag{12}$$

where h is a nonlinear function that models the sensor measurements. Now, to linearize the equations that are nonlinear, the states are converted by the following model given by

$$\delta x(t) \triangleq x(t) - \hat{x}(t), \tag{13}$$

where $\delta x(t)$ is the discrepancy between the true state vector and the state estimate. Since the equations are nonlinear and we still need to propagate them forward in time, the EKF integrates the non-linear equations over the difference in time by

$$\hat{x}_{t+1}^{-} = \int_{t_{t+1}}^{t_t} f[x(t), u(t), t]dt \; + \hat{x}_t^{+} \tag{14}$$

then the following linear dynamics model is

$$F_t = \frac{\delta f}{\delta x}|_{\hat{x}_t^{+}} \tag{15}$$

Then, to update the estimates of the states utilizing the sensor measurements as they come in which may be non-linear, we start by predicting the measurement using the measurement model equation along with the most recent measurement with

$$\hat{z}_t = h(\hat{x}_t^{-}, t_t) \tag{16}$$

and

$$\delta z_t = z_t = \hat{z}_t. \tag{17}$$

δz_t is the pre-update measurement residual that describes the difference between the predicted value and true measurement. We can now linearize the nonlinear measurement function h

$$H_t = \frac{\delta h}{\delta x}|_{\hat{x}_t^-} \tag{18}$$

where H being the linearized matrix of equations. Then, the measurement update equation can be reduced to the following:

$$\delta \hat{x}_t^+ = K_t \delta z_t \tag{19}$$

This final equation shows how the perturbation state is updated as the filter progresses through time. This state starts at zero and, during each iteration of the filter, is updated then finally reset to zero after each iteration of the Kalman filter. The perturbation state is added to the state to produce the final, full state estimate.

2.3 Covariance and Monte Carlo Analyses

Currently, Monte Carlo analysis is a very common tool for trade-space analysis. However, it requires hundreds, and often thousands, of simulated runs to obtain the required results. Since the Monte Carlo method obtains its results, the statistics of the system by averaging all the runs conducted, the more runs conducted the more accurate the results. This leads to a considerable computational load and a lengthy run-time. Monte Carlo analysis is used in a wide scope of scientific fields and is a versatile tool for nonlinear analysis. In the specific case of a GNC system, a Monte Carlo simulates the sensors with the given specifications, computes the covariance and error states by generating a large amount of samples of the trajectory and measurements, and then estimates the states and covariances. In the following Figure (Fig. 16), we can see a flow diagram of a Monte Carlo simulation for navigation.

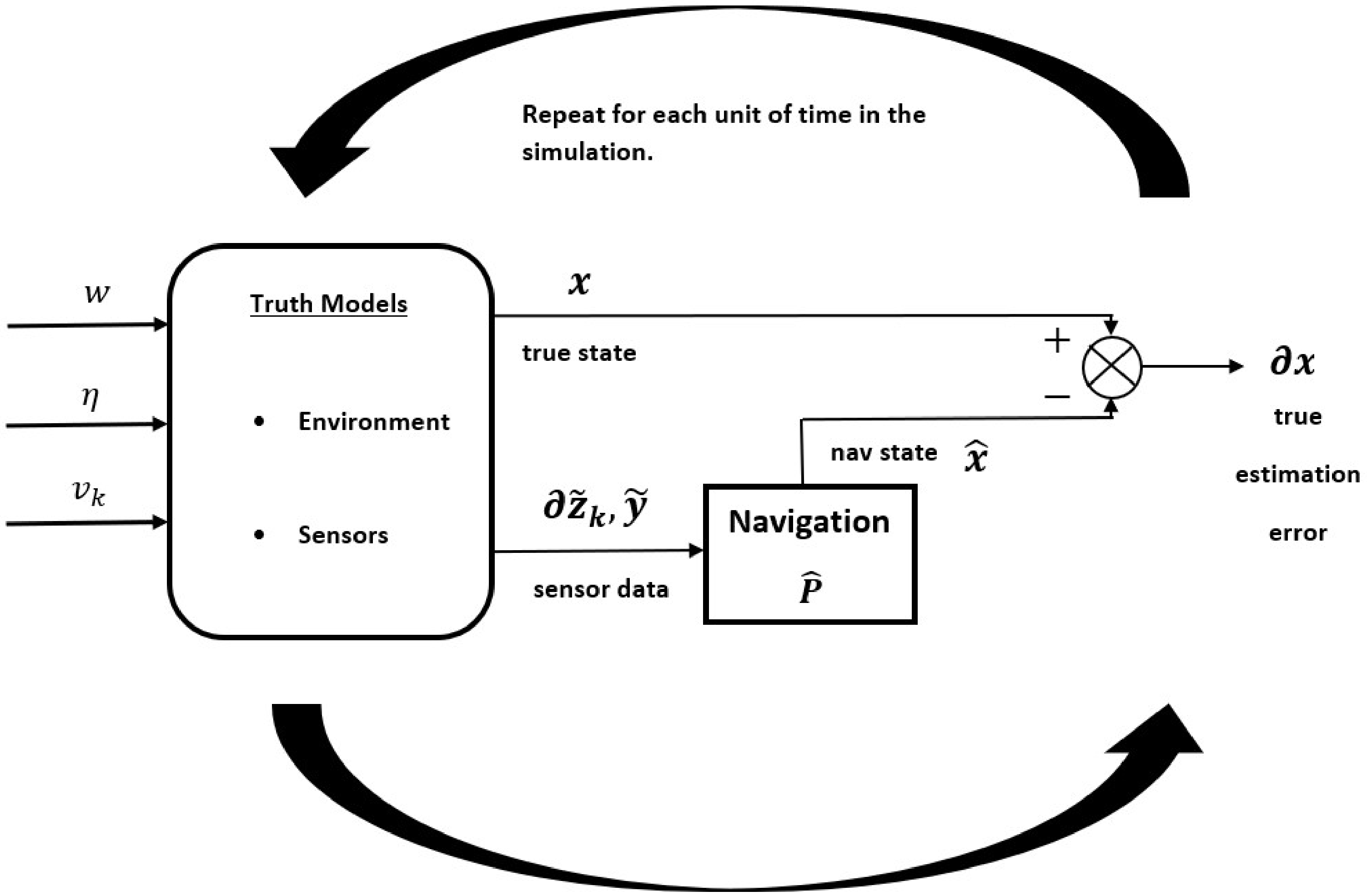


Figure 2: Monte Carlo simulation for a generalized GNC problem. This loop is ran for each time-step in the trajectory till one has completed the simulation. This represents a single estimate, with hundreds or thousands of these estimates being required. Then, the statistics of the errors in the system are the final outputs. The process noise w, continuous sensor noise η (e.g. the inertial measurement unit (IMU)), and discrete sensor noise v_k form the dynamics for the truth state x. The simulated sensor data is defined as $\delta\tilde{z}_k$, which are discrete measurements by sensors, and $\tilde{y}$, which are continuous IMU data. Finally, δx is the difference between the true state and the estimated state. The output value is δx. The mean value obtained from the Monte Carlo simulation is a statistical approximation of the true value. The statistical sample can be improved by increasing the total number of Monte Carlo runs.

2.3.1 Covariance Analysis

Covariance analysis can provide the covariance of the system just like a Monte Carlo but in a single simulation run. This eliminates the computational burden that comes with the Monte Carlo analysis. Covariance analysis has a few extra requirements that are critical to keep in mind. Covariance analysis requires that the navigation equations be linearized or linearly approximated. Therefore, if the equations at hand induce many non-linearities that cannot be linearly approximated, a Monte Carlo may be the only option. One must explicitly define the structure, the truth model, and the design model for which the filter is based [6]. Also, all errors must be Gaussian distributed. Since all errors must be Gaussian,the equations linear, and our filter is being fed the truth of the system, the covariance relationships are independent of our estimate [6]. This allows us to obtain the covariance, and thus the distribution information, of the simulation without the estimate. The following equations show how we arrive at the covariance without the estimate and assume a full Kalman filter, which was defined above. $P_e(t)$ is the covariance of the estimation errors committed by the given filter and $P_a(t)$ is the covariance of the augmented state. The augmented state is a the state and errors of the states augmented together. Since all processes are assumed to be zero mean, we can define $P_a(t)$ and $P_e(t)$ as

$$P_a(t) = E\{x_a(t)x_a^T(t)\}, \tag{20}$$

$$P_e(t) = E\{e_t(t)e_t^T(t)\}, \tag{21}$$

where e_t is the error committed by the filter and x_a being the augmented state vector at the given time. Propagating between samples is done by integrating

$$\dot{P}_a(t) = F_a(t)P_a(t) + P_a(t)F_a^T(t) + G_a(t)Q_t(t)G_a^T(t), \tag{22}$$

with a linear system with structure defined as defined as F_a and G_a. With uncertainty defined as Q_t The measurement update can be derived as

$$P_a(t_i^+) = A_a(t_i)P_a(t_i^-)A_a^T(t_i) + K_a(t_i)R_t(t_i)K_a^T(t_i), \tag{23}$$

where our initial covariance is defined as

$$P_a(t_0) = \begin{bmatrix} -P_{t0} & 0 \\ 0 & P_0 \end{bmatrix} \tag{24}$$

we then define the follow equations

$$A_a(t_i) = \begin{bmatrix} I & 0 \\ K(t_i)H_t(t_i) & [I - K(t_i)H(t_i)] \end{bmatrix}, K_a(t_i) = \begin{bmatrix} 0 \\ K(t_i) \end{bmatrix}, \tag{25}$$

A_a is required to update our covariance analysis equation and K_a being the Kalman gain of our estimate with zeros added on top.

Finally the desired error covariance can be obtained with

$$P_e(t) = C_a(t)P_a(t)C_a^T(t). \tag{26}$$

where

$$C_a(t) = \begin{bmatrix} -C_t(t) & C(t) \end{bmatrix} \tag{27}$$

. C is the critical states or quantities we care about. If we are only interested with a certain number of states in our system using C we can only return the ones that we are interested in. If we are interested in all of the states $C(t) = I$. We end up with the covariance of the state using the sensor covariance values and without having to calculate individual estimates. This saves a large amount of computation time.

2.4 Modular and Pluggable Navigation Architectures

2.4.1 Scorpion

Scorpion is the precursor to the Bayesian estimation software we have defined in this thesis as Position, Navigation, and Timing Operating System (pntOS) [4]. Scorpion was created at the Air Force Institute of Technology (AFIT) Autonomy and Navigation Technology Center (ANT Center) and was initially designed to be a support tool for academic research and was built to run on a Java Virtual Machine (JVM). It was written primarily in Kotlin. This tool was created to be a pluggable software architecture but had some shortfalls in the difficulty for users to create their own functions alongside the government-owned components. Scorpion did not have a large software architecture like its successor pntOS, which allows pntOS to create its own stand-alone programs.

2.4.2 pntOS

pntOS is a pluggable Bayesian estimation application that allows a software framework to generate navigation simulations and scenarios in a virtual environment. A library exists alongside pntOS, called Navigation Toolkit, that allows for users to utilize modular Bayesian estimators and assemble plugins that are compatible with pntOS. The overall architecture creates an environment where users can create different filters, sensors, and algorithms that can be easily accessed and used in a wide variety of applications. pntOS is a government-owned, fully pluggable architecture for building navigation systems. pntOS is designed so that a system can be created from a mixture of proprietary and government-owned components. It defines a much more user-friendly architecture where users can easily create their own tools and, within the framework, will work with any other user-created tools. pntOS is analogous to an operating system in the sense that, like a computer operating system manages basic computer functions like task scheduling and executing programs, pntOS manages the basic functions in a Position Navigation and Timing (PNT) system. The majority of the work in this thesis is conducted inside of the reference library of pre-made filters, models, and algorithms that are able to be used to create viper plugins inside ofpntOS known as the Navigation Toolkit. pntOS can be split into 3 main parts: the pntOS architecture, navigation toolkit, and viper plugins.

2.4.2.1 Architecture

The main inspiration for the rework of Scorpion to create pntOS was to create an architecture that can be operationalized. pntOS is just the architecture itself. It creates an environment to allow various sensors, perform data fusion or filtering, and produce a navigation solution. The goal is for the architecture to be easy to follow and to develop inside of. This is to allow other parties to create their own software that conforms to the pntOS architecture and have a centralized environment for all navigation research and operationalized development.

2.4.2.2 Viper Plugins

Viper plugins are government provided pre-built reference plugins that plug into pntOS. These are tools that can be plugged into the pntOS architecture to do various different tasks. Plugins can be many different things, such as models, sensors, algorithms, etc. Figure 3 shows a basic sensor plugin example.

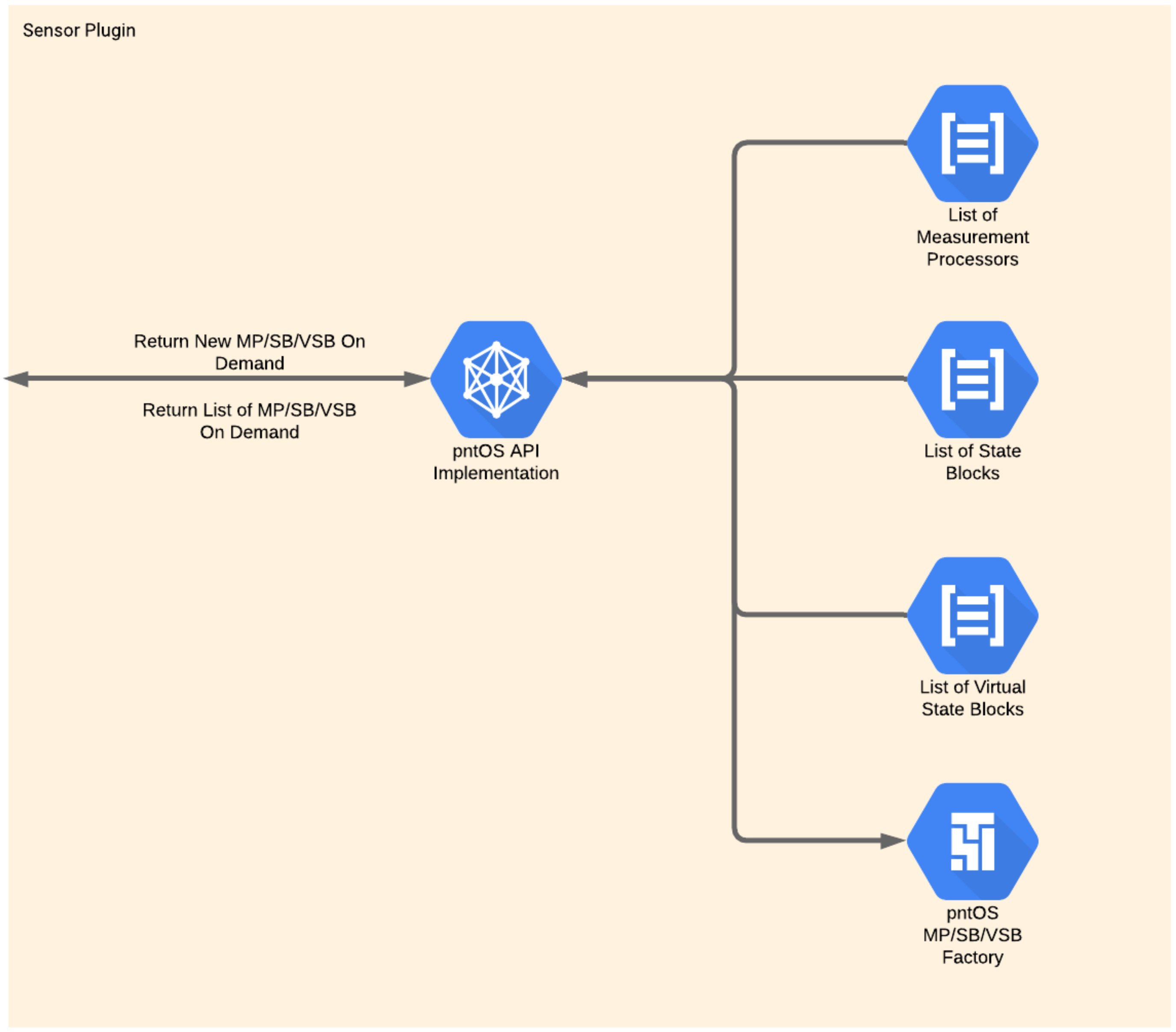

Sensor Plugin
Return New MP/SB/VSB On Demand
Return List of MP/SB/VSB On Demand
pntOS API Implementation
List of Measurement Processors
List of State Blocks
List of Virtual State Blocks
pntOS MP/SB/VSB Factory

Figure 3: Diagram of a Basic Sensor Plugin [3]. Stateblocks and measurement processors are used in coordination to create a full navigation simulation.

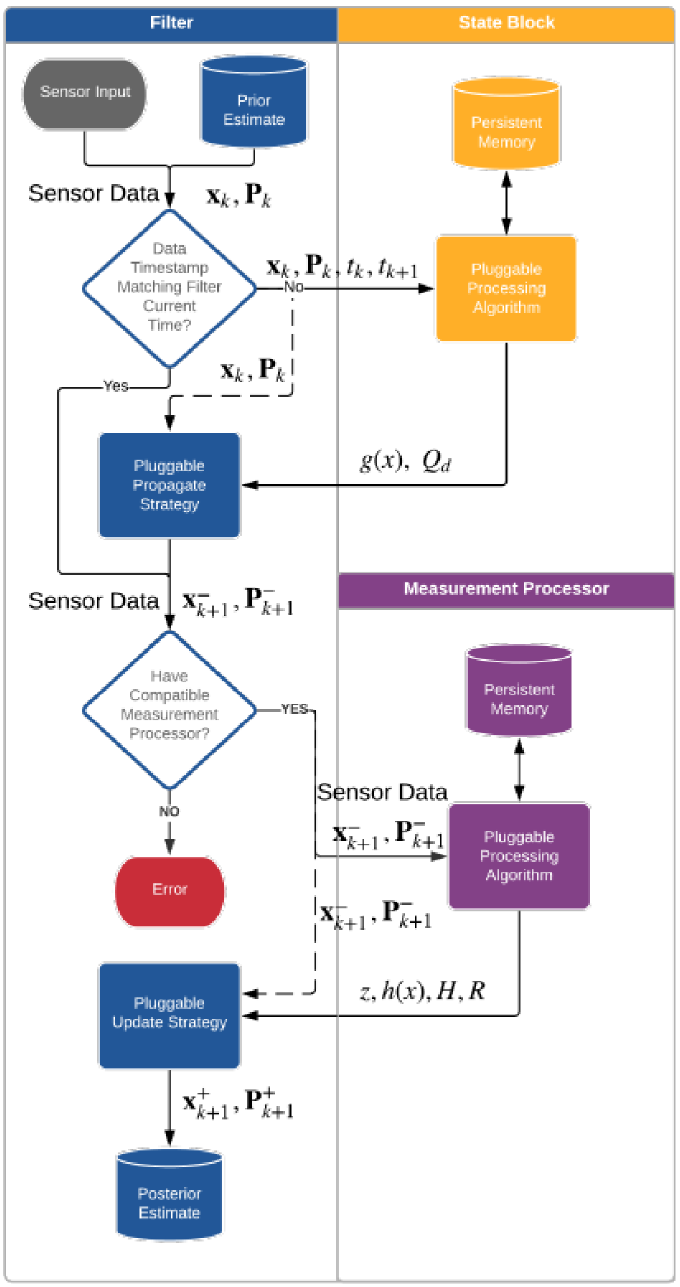
2.4.3 Navigation Toolkit

Navigation Toolkit is a collection of government provided prepackaged algorithms, filters, and models within a library. These are prepackaged tools that can be utilized in pntOS to create plugins for pntOS. A full decomposition of this can be seen in Figure 19. For example, the simulations we will explain later in Chapter III and Chapter IV will use the Pinson15 model and Extended Kalman Filter. Both of these are prepackaged in the navigation toolkit. This software setup allows researchers to easily modify sensor values, add and remove sensors, as well as swap filters. The main three goals of this software library effort known as navigation toolkit are modular algorithms, pluggable filters, and pluggable sensors. Now the parts of the navigation toolkit that make up a simulation will be described below.

State Blocks A state block represents a collection of states whose propagation is not dependent on any states outside of the given state block [4]. State blocks are a set of N states, that house the info required to propagate the states forward in time.

Measurement Processors A measurement processor represents the relationship of a measurement to the state vector. Given a sensor measurement, the measurement processor provides the fusion engine with a model to update the states.

Fusion Engines Finally the Fusion engine brings everything together, the stateblock and measurement-processor, then produces the full estimates of all the states that are required by the state blocks. The estimates are created for the past, present, and future of the states. The fusion engine operates by the measurement processor passing the engine the measurements then the models from the state blocks and measurement processor go to the fusion strategy.



Filter
State Block
Sensor Input
Prior Estimate
Persistent Memory
Sensor Data
$\mathbf{x}_k, \mathbf{P}_k$
Data Timestamp Matching Filter Current Time?
$\mathbf{x}_k, \mathbf{P}_k, t_k, t_{k+1}$
No
Pluggable Processing Algorithm
Yes
$\mathbf{x}_k, \mathbf{P}_k$
Pluggable Propagate Strategy
$g(x),\ Q_d$
Measurement Processor
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Have Compatible Measurement Processor?
YES
Persistent Memory
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Pluggable Processing Algorithm
NO
Error
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Pluggable Update Strategy
$z, h(x), H, R$
$\mathbf{x}^+_{k+1}, \mathbf{P}^+_{k+1}$
Posterior Estimate

Figure 4: Diagram of the Viper Framework [4]. Stateblocks, measurement processors, and stateblocks make up everything you need for a full navigation simulation.

2.4.4 Pinson15 Error Model

For the analysis, we use an out-of-the-box Pinson15Ned state block. This pinson15 state block updates as the filter runs and what happens inside of them will be described in detail in their own respective sections. The measurement processor houses a model of the sensor you have called and instructions on how the measurement will be included in your used state blocks. A pinson15 is a set of 15 states bundled together that model the errors of an Inertial Navigation System (INS) in a moving North East Down (navigation) frame. A Pinson15 requires an initial value for the 15 states used and we set this up with the initial sigma values of the given sensors as well as the initial bias in the gyro and accelerometer. A Pinson15 model can be described as the following [15]

$$X = \begin{bmatrix} \delta p_n \ \delta p_e \ \delta p_{vert} \ \delta v_n \ \delta v_e \ \delta v_d \ \epsilon_n \ \epsilon_e \ \epsilon_d \ b_{a_x} \ b_{a_y} \ b_{a_z} \ b_{g_x} \ b_{g_y} \ b_{g_z} \end{bmatrix}^T \tag{28}$$

where δp is north-east-down position errors, δv represents NED velocity errors, η are the tilt errors about the NED axis, b_a x,y,z are accelerometer time-correlated biases, and b_g x,y,z gyro time-correlated biases. The linearized dynamics model can be expressed as

$$\begin{bmatrix} \dot{\delta p} \\ \dot{\delta v} \\ \dot{\delta \epsilon} \\ \dot{b}_a \\ \dot{b}_g \end{bmatrix} = \begin{bmatrix} F_{pp} & F_{pv} & F_{p\epsilon} & 0_{3\times3} & 0_{3\times3} \\ F_{vp} & F_{vv} & F_{v\epsilon} & C_s^n & 0_{3\times3} \\ F_{\epsilon p} & F_{\epsilon v} & F_{\epsilon\epsilon} & 0_{3\times3} & C_s^n \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{aa} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{gg} \end{bmatrix} \begin{bmatrix} \delta p \\ \delta v \\ \delta \epsilon \\ b_a \\ b_g \end{bmatrix} \tag{29}$$

Before we define the submatrices we will define the variables used inside of them with the following table where the sub-matrices are defined as the following

Table 1: Variable Definitions for Pinson 15 Error Model

Variable	Description
L	Latitude
v_n	North Velocity
v_e	East Velocity
v_d	Down Velocity
R_e	Earth Radius
f_n	Measured Specific Force North Direction
f_e	Measured Specific Force East Direction
f_d	Measured Specific Force Down Direction
Ω	Earth Rotation Rate

$$F_{pp} = \begin{bmatrix} 0 & 0 & \frac{-v_n}{R_e^2} \\ \frac{v_e tanL}{R_e cosL} & 0 & \frac{v_e}{R_e^2 cosL} \\ 0 & 0 & 0 \end{bmatrix} \tag{30}$$

$$F_{pv} = \begin{bmatrix} \frac{1}{R_e} & 0 & 0 \\ 0 & \frac{1}{R_e cosL} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{31}$$

$$F_{p\epsilon} = 0_{3\times3} \tag{32}$$

$$F_{pv} = \begin{bmatrix} v_e(2 * \Omega cosL \frac{v_e}{R_e cos^2 L}) & 0 & \frac{v_e^2 tanL - v_n * v_d}{R_e^2} \\ 2\Omega(v_n cosL - v_d sinL) + \frac{v_n v_e}{R_e cos^2 L} & 0 & \frac{-v_e(v_n tanL + v_d)}{R_e^2} \\ 2\Omega v_e sinL & 0 & \frac{v_n^2 + v_e^2}{R_e^2} \end{bmatrix} \tag{33}$$

$$F_{vv} = \begin{bmatrix} \frac{v_d}{R_e} & -2(\Omega sinL + \frac{v_e}{R_e cos^2 L}) & \frac{v_n}{R_e} \\ 2\Omega sinL + \frac{v_e tanL}{R_e} & \frac{v_n tanL + v_d}{R_e} & 2\Omega cosL + \frac{v_e}{R_e} \\ \frac{-2v_n}{R_e} & -2(\Omega cosL + \frac{v_e}{R_e}) & 0 \end{bmatrix} \tag{34}$$

$$F_{v\epsilon} = \begin{bmatrix} 0 & -f_d & f_e \\ f_d & 0 & -f_n \\ -f_e & f_n & 0 \end{bmatrix} \tag{35}$$

$$F_{\epsilon p} = \begin{bmatrix} -\Omega sinL & 0 & \frac{-v_e}{R_e^2} \\ 0 & 0 & \frac{v_n}{R_e^2} \\ -\Omega cosL - \frac{v_e}{R_e cos^2 L} & 0 & \frac{v_e tanL}{R_e^2} \end{bmatrix} \tag{36}$$

$$F_{\epsilon v} = \begin{bmatrix} 0 & \frac{1}{R_e} & 0 \\ -\frac{1}{R_e} & 0 & 0 \\ 0 & \frac{tanL}{R_e} & 0 \end{bmatrix} \tag{37}$$

$$F_{\epsilon\epsilon} = \begin{bmatrix} 0 & -\Omega sinL + \frac{v_e}{R_e tanL} & \frac{v_n}{R_e} \\ \Omega sinL + \frac{v_e tanL}{R_e} & 0 & \Omega cosL + \frac{v_e}{R_e} \\ \frac{-v_n}{R_e} & -\Omega cosL - \frac{v_e}{R_e} & 0 \end{bmatrix} \tag{38}$$

where C_s^n is a direction cosine matrix, which takes a vector expressed in the body frame and expresses that vector in the navigation frame. F_{aa} describes the decaying exponential terms of the first order Gauss-Markov process given by

$$F_{aa} = \begin{bmatrix} \frac{1}{\tau_a} & 0 & 0 \\ 0 & \frac{1}{\tau_a} & 0 \\ 0 & 0 & \frac{1}{\tau_a} \end{bmatrix} \tag{39}$$

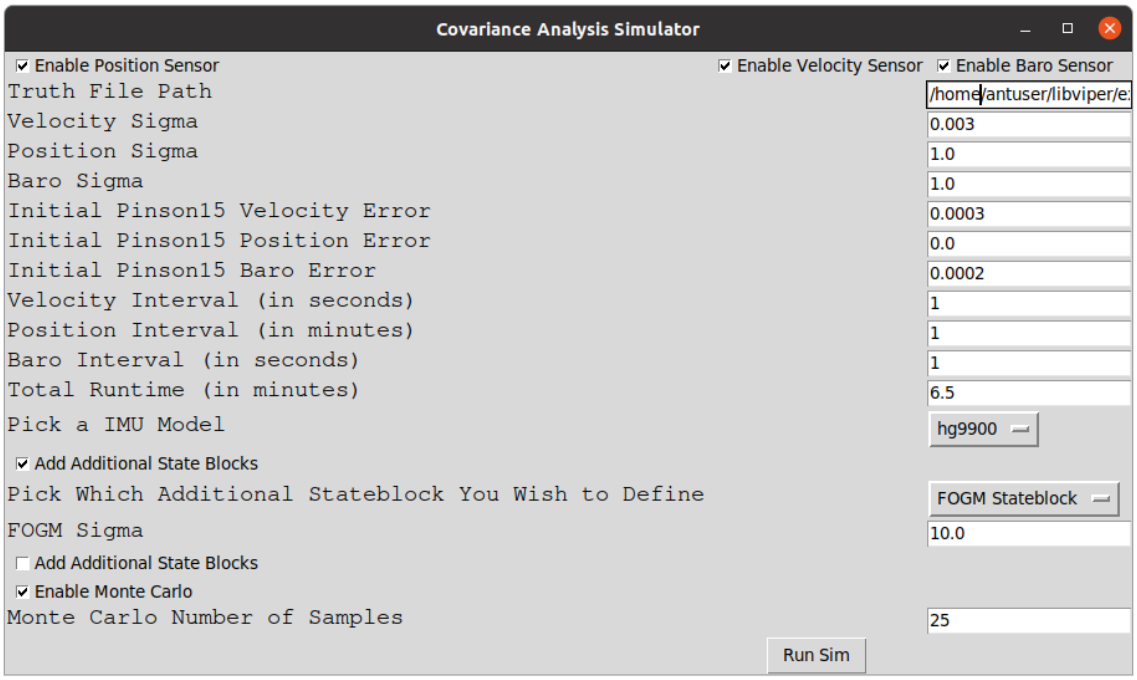
where τ_g is a time constant of the accelerometer time-correlated biases. F_{gg} is the same, but for the time constant of the gyro time correlated biases given as

$$F_{gg} = \begin{bmatrix} \frac{1}{\tau_g} & 0 & 0 \\ 0 & \frac{1}{\tau_g} & 0 \\ 0 & 0 & \frac{1}{\tau_g} \end{bmatrix}. \tag{40}$$

2.5 Covariance Analysis Tool

The covariance analysis simulation tool will be built on over time to add more and more different sensors, stateblocks, and ease of usability. Currently, there exists a headless C++ version that is wrapped in python bindings and a UI version that allows users to quickly use the tool without any knowledge of how to use pntOS and navigation toolkit. The User Interface (UI) version is more limited as you have to use the stateblocks, sensors, and models already added by selecting different check boxes or drop downs as you can see an example of the UI in (Fig. 20). This helps users that are not as accustomed to the tool or users who simply want to quickly run a simulation and get plotted results. A user can easily run the covariance simulation program, pick their IMU/sensors, pick the different stateblocks they wish to run, give the path to their truth data Comma-Separated Values (CSV), and give all the relevant sigmas/intervals for the sensors.

The simulation using the navigation toolkit takes in a CSV or zip of the truth trajectory. CSV for the created tool UI or coded covariance simulator. A zip is used when running the simulation with Magnetic Anomaly Navigation (MagNav). The MagNav version of this tool is inside of viper-MagNav which is a different branch of libviper. When using MagNav measurements we have to use a certain truth that includes the magnetic anomaly maps. Since the simulation is taking in the truth and creating the simulated flight we have to create our own measurements. The errors in the measurements of the different sensors is created by adding noise to the truth measurements given the sigma values passed in by the user. The filter runs with an extended Kalman Filter (EKF), updates the state block models, and creates simulated sensor measurements for all time intervals. Our covariance analysis as described above has a few requirements that are easily met with the EKF. The errors must be Gaussian and the navigation equations must be linearized.


Covariance Analysis Simulator

☑ Enable Position Sensor ☑ Enable Velocity Sensor ☑ Enable Baro Sensor

Truth File Path	/home/antuser/libviper/e:
Velocity Sigma	0.003
Position Sigma	1.0
Baro Sigma	1.0
Initial Pinson15 Velocity Error	0.0003
Initial Pinson15 Position Error	0.0
Initial Pinson15 Baro Error	0.0002
Velocity Interval (in seconds)	1
Position Interval (in minutes)	1
Baro Interval (in seconds)	1
Total Runtime (in minutes)	6.5
Pick a IMU Model	hg9900

☑ Add Additional State Blocks

Pick Which Additional Stateblock You Wish to Define	FOGM Stateblock
FOGM Sigma	10.0

☐ Add Additional State Blocks

☑ Enable Monte Carlo

Monte Carlo Number of Samples	25

Figure 5: Example of UI ran Straight Flight Example using Pinson15 model.

If users want to create their own stateblocks, their own custom IMUs, or further customize the tool, the terminal ran version is what they would use. By using one the many examples libviper offers as a template. Users can easily set the values of their own stateblocks to add custom models or can call on the custom IMU and set their own sigma values. The "straight_flight_example" view from a IDE can be seen in appendix A. This shows where users can add their own custom IMUs, edit sigma values, enable and disable sensors, set where the truth data for the simulation is coming from and edit the interval at which sensors are updated.

2.6 Related Work

Most covariance analysis related papers take a single navigation scenario and use covariance analysis to explain how the navigation solution is valid. An example would be a lunar lander using covariance analysis to show the solution and the math creates our true states for landing a rover on the moon. The covariance papers [7, 16, 17, 18] explain the math behind the filtering to model the flight and then show the math on how the states of the system can be modeled with a covariance analysis. These papers focus more on the navigation problem at hand and use covariance analysis as a way to show how a solution can be found in a simulation environment. Of course, covariance analysis works great with space related navigation since testing these scenarios in the real world are close to impossible or abundantly costly.

A paper by Christensen and Geller [7] proposes that given a covariance analysis in a closed loop navigation solution will achieve similar results and is a much more computationally cost effective solution. The authors show that given a closed-loop guidance scenario (rocket-sled) using Monte-Carlo simulation the run time was over 26,000 hours. Given a linear covariance, you can achieve the covariance of the system and only have a 38 hour run time. The authors go on to explain that their simulation takes into account many aspects that would be crucial to closed loop navigation. Such as error budgets, effect of changes to navigation/control scheme, and effect of sensor specifications and interference levels. With these items taken into account it can be shown that not only will linear covariance give the results you want within 3 sigma of error but also you can expect all the usual helpful data you would expect from a full Monte Carlo run. This is one of few papers that have a goal to explain covariance analysis with a simple example. Most papers will utilize covariance analysis to analyze their equations and show that their navigation scenario is valid. This closed loop navigation paper by Geller and Christensen sets out to explain what covariance analysis is and how it can be a valuable trade space analysis tool that does not require extensive computation time like that of a Monte Carlo analysis.

III. Scholarly Article: Validation of Doppler Lidar Sensor using Covariance Analysis

3.1 Abstract

A novel Doppler Light Detection And Ranging (LiDAR) sensor has been developed with the help of NASA technology transfer. This sensor can make highly accurate, three-dimensional velocity measurements in the body frame of a vehicle. However, the practicability of such a sensor in an inertial measurement unit (IMU)-mechanized navigation solution is unknown. A covariance analysis can be run with a simulation flight across multiple trajectories as well as different IMU grades to understand the sensors accuracy and contribution to a navigation system. With a full simulated analysis of this sensor, it is much easier for researchers and others to understand the full benefits that a Doppler LiDAR system might bring to a navigation solution. We conduct the analysis by way of covariance analysis as well as Monte Carlo and give a brief explanation of what covariance analysis is and how it can be used to obtain results for this new sensor. We define this new software tool we have created and speak on how researchers can use it to conduct covariance analysis tests of their own.

3.2 Introduction

In this paper we document the results of analysing the navigation performance of a newly developed Doppler LiDAR sensor using a technique known as covariance analysis [7]. The tool used in conducting the analysis is built within a modular multi-source navigation sensor fusion library [4]. Many papers utilize covariance analysis for different individual analysis problems such as closed loop navigation[7], space rendezvous[1], and lunar landers [2]. Currently, most covariance analysis research defines a specific problem to be researched and shows the results of said experiment using covariance analysis. We will be using covariance analysis to define a specific experiment with the results of the Doppler LiDAR sensor. We also will be defining the software package used to create this experiment and how other researchers can utilize this software to create their own experiments.

The creation of a covariance analysis tool inside of a modular software package will allow researchers to plug-and-play their own data and run covariance analysis experiments of their own. Section 4.2 characterizes Monte Carlo analysis as well as covariance analysis and gives a brief introduction to trade-space analysis. Section 4.3 defines the software package used and describes how the software package is modified to allow the creation of covariance analysis simulations. Section 4.4 defines the specific experiment we conducted to prove the created covariance analysis simulation works as intended and show the capability of the sensor. Finally, section 3.6 summarizes the contents of the effort.

3.3 Background

3.3.1 Trade-Space Analysis

The research of alternative or complementary navigation is the search for new and more reliable or accurate ways to navigate the ever-changing environment. It comes as no surprise that trade-space analysis is a considerable tool for alternative navigation research. The evaluation of sensor pairings, error budgets, and disturbances are all navigation-related trade-space analysis problems. For example, taking an IMU sensor paired with a velocity sensor to experiment with positioning accuracy is an example of trade-space analysis in navigation. Taking a Global Positioning System (GPS) sensor and measuring the error that is allowed before the position solution progresses beyond the error threshold would be a good example. How a navigation system would respond to an interruption in GPS over a set amount of time is another trade-space analysis problem. Executing tests to research sensor pairings, error budgets, and disturbances in the real world, given the broadness of the experiment, can be extremely expensive and time consuming. Creating simulations for trade-space experiments and running a simulated flight is a much more practical way to conduct these experiments and can provide potentially similar statistical data to what a real test would provide. A covariance analysis on a simulation of an example flight with all parameters for the given sensors set prior to the run can give you results on sensor pairings, error budgets, and disturbances quickly and efficiently.

3.3.2 Monte Carlo Analysis

Currently, Monte Carlo analysis is the most common tool for trade-space analysis. However, it requires hundreds, and often thousands, of simulated runs to obtain the required results. This leads to a considerable computational load and a lengthy run-time. Monte Carlo analysis is used in a wide scope of scientific fields and is a versatile tool. In our specific case in a Guidance, Navigation, and Control (GNC) system, a Monte Carlo simulates the sensors with the given specifications, computes the covariance and error states by generating a large amount of samples of each state, and then estimates the covariances.

In the following Figure 16 we can see a flow diagram of a Monte Carlo simulation for navigation.

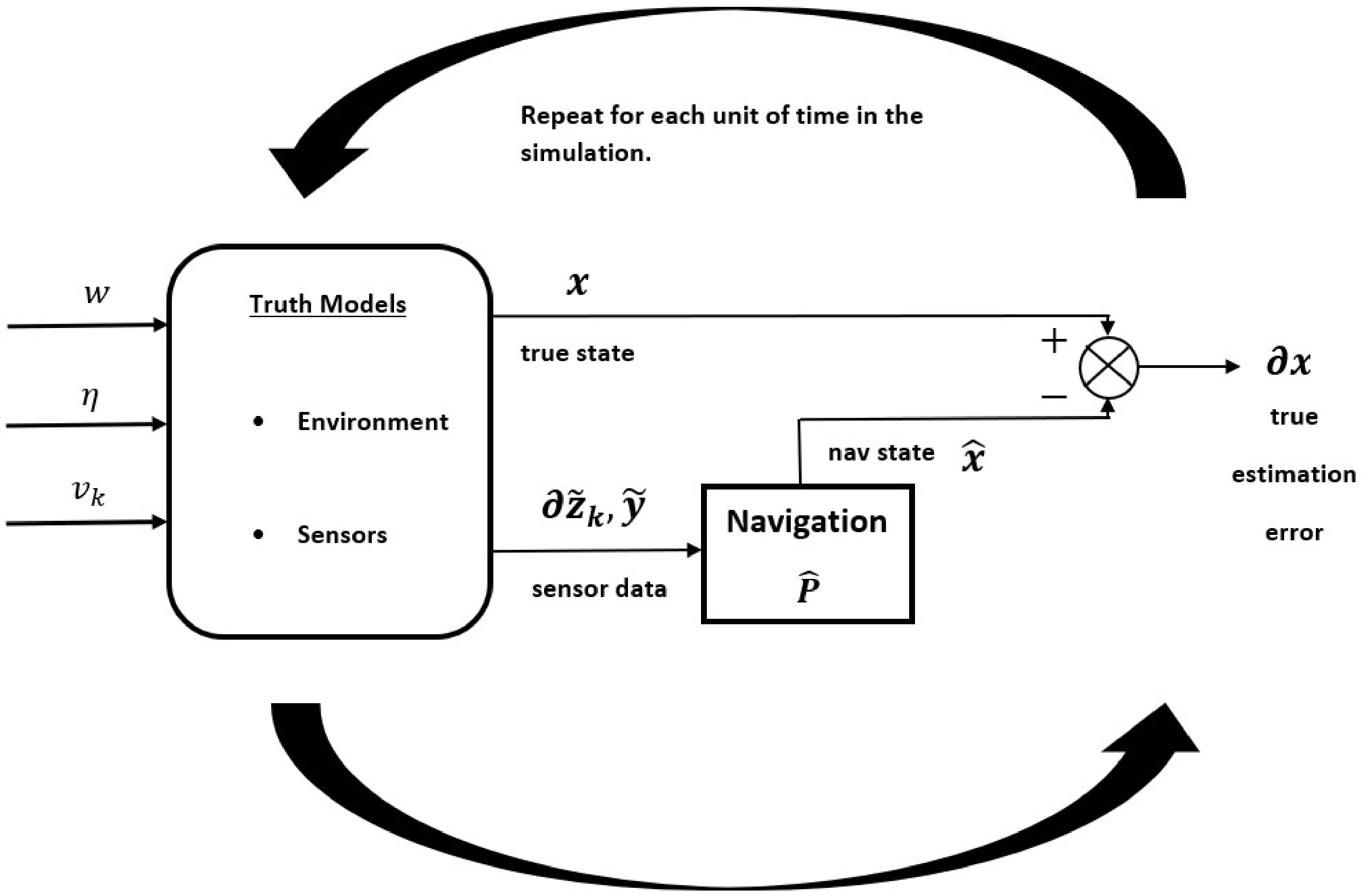


Figure 6: Monte Carlo simulation for a generalized GNC problem.

The w process noise, η continuous sensor noise, and v_k discrete sensor noise form our dynamics for the truth model to form our true state x. The simulated sensor data is defined as $\delta\tilde{z_k}$, which are discrete measurements by sensors, and $\tilde{y}$, which are continuous IMU data. Finally, δx is the difference between the true state and estimated state. The output value is δx. The mean value we get from our Monte Carlo simulation is a statistical approximation to the true value. You can improve the statistical sample by increasing the total amount of Monte Carlo runs.

3.3.3 Covariance Analysis

Covariance analysis can provide similar statistical data to a Monte Carlo but in a single simulation run. This eliminates the computational burden that comes with the Monte Carlo analysis. Covariance analysis has a few extra requirements that are critical to keep in mind. Covariance analysis requires that the truth model be linearized or linearly approximated. Therefore, if the situation induces many nonlinearities, a Monte Carlo may be the only option. One must explicitly define the structure and uncertainties of the truth model and the design model for which the filter is based [6]. Also, all errors must be Guassian. Since all errors must be Guassian and the equations linear, the covariance relationships are independent of the measurement time history [6]. We follow the covariance analysis as outlined in [7][19][17].

3.4 Software Description

3.4.1 Navigation Toolkit

Navigation toolkit is a modular Bayesian estimation software library designed to assist users in the creation of navigation filters. Navigation toolkit exposes an API for developing pluggable filter components and is an operationalized version of Scorpion

[?]. This software setup allows researchers to easily modify sensor values, add and remove sensor suites, as well as swap filters.

Navigation toolkit works by way of fusion engines/strategy and state blocks, a full decomposition of this can be seen in Figure 19. The fusion engine/stategy sets up the users filter of choice. As of now the options out of the box are an unscented Kalman Filter and a Extended Kalman Filter. For our example as mentioned earlier, we are going to use an EKF since this works with our covariance analysis requirements well. Next, state blocks are a set of N states, these state blocks house the info required to propagate the states forward in time. For our analysis we use an out of the box Pinson15Ned state block, which is a set of 15 state blocks bundled together that model the errors of an INS in the North East Down frame. A Pinson15 requires an initial value for the 15 states used and we set this up with the initial sigma values of our given sensors as well as the initial bias in the gyro and accelerometer.

Now, we have a full state block loaded and the initial conditions set. One more item is needed for the Pinson15 to propagate our solution. Since our Inertial Navigation System (INS) errors change as a function of the vehicle's motion, we have to give the Pinson model the approximate position, rotation, velocity, and specific force. For our covariance analysis this is where our truth is fed in. Instead of an "approximation" we will use the truth of the flight for this part. Every propagation the Comma-Separated Values (CSV) will be called again to feed in the current true position of the vehicle into the model instead of feeding in the approximate location output by the extended Kalman Filter (EKF). With all of this we have a full simulation that follows the rules of covariance analysis and will give us a accurate covariance of the flight in a singular run.

The ease of adding, editing, and deleting different sensors, using different filters, and utilizing different models lets us analyze this Doppler LiDAR sensor to an extent unreachable before. Navigation toolkit exposes a framework that researchers can easily write their own models or filters in and have access to other researchers created filters/models. The creation of navigation toolkit was to allow researchers to work on navigation simulations or build sensors in a common place where other researchers can pull what they have done and add it to their work without having to reinvent the wheel.

A Pinson15 model can be described as the following:

$$X = \begin{bmatrix} \delta p_n \\ \delta p_e \\ \delta p_{vert} \\ \delta v_n \\ \delta v_e \\ \delta v_d \\ \epsilon_n \\ \epsilon_e \\ \epsilon_d \\ b_{a_x} \\ b_{a_y} \\ b_{a_z} \\ b_{g_x} \\ b_{g_y} \\ b_{g_z} \end{bmatrix} \tag{41}$$

where δp is n-e-vertical position errors, δv represents NED velocity errors, η about the NED axis, b_a x,y,z accelerometer time correlated bias, and b_g x,y,z gyro time correlated biases.

The linearized dynamics model can be expressed as:

$$\begin{bmatrix} \dot{\delta p} \\ \dot{\delta v} \\ \dot{\delta \epsilon} \\ \dot{b}_a \\ \dot{b}_g \end{bmatrix} = \begin{bmatrix} F_{pp} & F_{pv} & F_{p\epsilon} & 0_{3\times3} & 0_{3\times3} \\ F_{vp} & F_{vv} & F_{v\epsilon} & C_s^n & 0_{3\times3} \\ F_{\epsilon p} & F_{\epsilon v} & F_{\epsilon\epsilon} & 0_{3\times3} & C_s^n \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{aa} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{gg} \end{bmatrix} \begin{bmatrix} \delta p \\ \delta v \\ \delta \epsilon \\ b_a \\ b_g \end{bmatrix} \tag{42}$$

where C_s^n is a direction cosine matrix, which rotates a vector from the accelerometer or and gyro sensor frame to the NED navigation frame. F_{aa} describes the decaying exponential terms of the first order Gauss-Markov process given by:

$$F_{aa} = \begin{bmatrix} \frac{1}{\tau_a} & 0 & 0 \\ 0 & \frac{1}{\tau_a} & 0 \\ 0 & 0 & \frac{1}{\tau_a} \end{bmatrix} \tag{43}$$

where τ_g is a time constant of the accelerometer time correlated biases. F_{gg} is the same, but for the time constant of the gyro time correlated biases given as:

$$F_{gg} = \begin{bmatrix} \frac{1}{\tau_g} & 0 & 0 \\ 0 & \frac{1}{\tau_g} & 0 \\ 0 & 0 & \frac{1}{\tau_g} \end{bmatrix} \tag{44}$$

Our simulation in navigation toolkit takes in a CSV of the flight's truth and truth of the filter, setting the values for the sensors such as sigmas, intervals, and picking an IMU model. The Pinson15 model requires the initial filter uncertainty, then the model can run the simulation propagating the EKF and outputting data. Our covariance analysis as described above has a few requirements that are easily met with our EKF and Pinson15 model. Our errors have to be Gaussian and our navigation equations have to be linearized. Since we are using a Pinson15 error model, we can assume the truth is zero for all time.

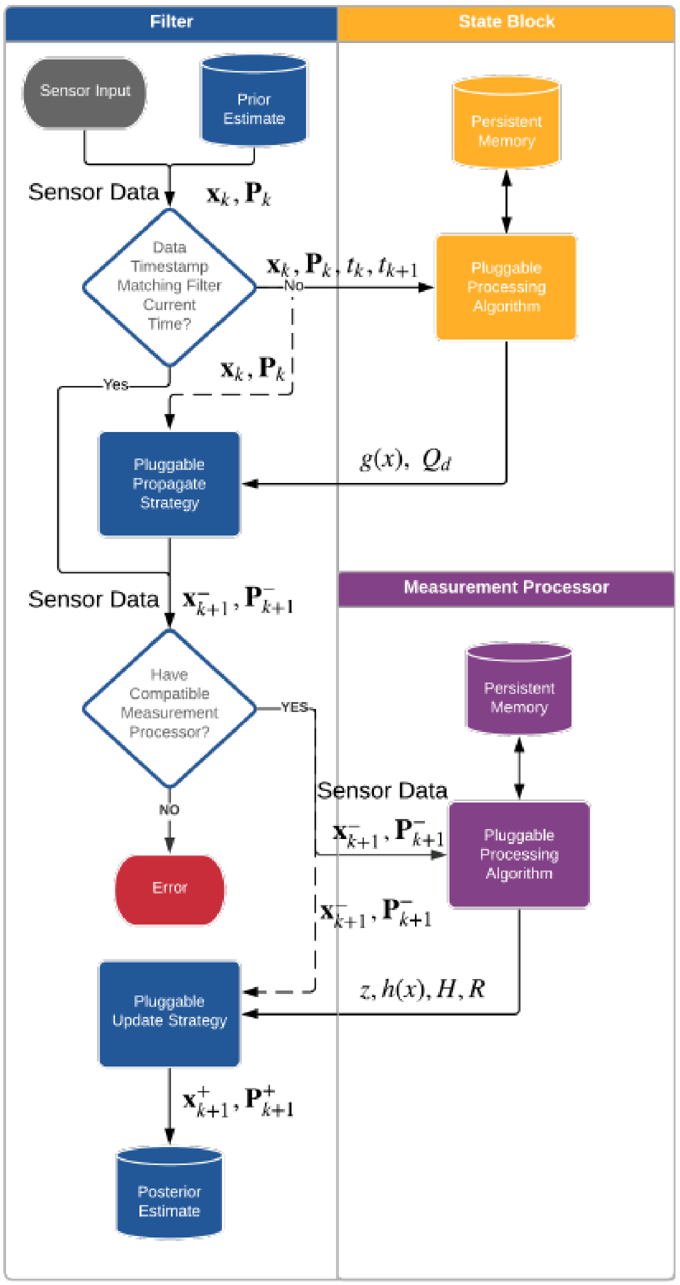

Filter
State Block
Sensor Input
Prior Estimate
Persistent Memory
Sensor Data
$\mathbf{x}_k, \mathbf{P}_k$
Data Timestamp Matching Filter Current Time?
$\mathbf{x}_k, \mathbf{P}_k, t_k, t_{k+1}$
No
Pluggable Processing Algorithm
$\mathbf{x}_k, \mathbf{P}_k$
Yes
Pluggable Propagate Strategy
$g(x),\ Q_d$
Measurement Processor
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Have Compatible Measurement Processor?
YES
Persistent Memory
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Pluggable Processing Algorithm
NO
Error
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Pluggable Update Strategy
$z, h(x), H, R$
$\mathbf{x}^+_{k+1}, \mathbf{P}^+_{k+1}$
Posterior Estimate

Figure 7: Diagram of the Navigation Toolkit Framework. [4]

3.5 Experiment

A novel Doppler LiDAR sensor has been developed with the help of NASA technology transfer. This sensor can make highly accurate, three-dimensional velocity measurements in the body frame of a vehicle. However, the practicability of such a sensor in an IMU-mechanized navigation solution is unknown. A covariance analysis is run with a simulation flight across multiple trajectories as well as different IMU grades to understand the sensors accuracy and potential contributions to a navigation system.

The vendor of the Doppler LiDAR sensor has provided us with error characteristics based on testing the sensor in the field. The navigation toolkit software package uses measurement processors to feed the measurement data into the model and filter. We will be using three measurement processors (velocity, baro, and position). These measurement processors simply need to be initialized and given the measurement covariance, and interval of data collects. Our values used for the measurement covariance and intervals as well as any other required parameters can be found in Table 5. The truth data used for this analysis is a CSV containing timing, latitude, longitude, height, and velocity in the body frame. The flight itself is a figure eight shown in Figure 8.

The simulation was conducted using the covariance analysis as well as running a Monte Carlo simulation of a thousand samples. For a baseline, first a simulation was run using covariance analysis with only a navigation IMU sensor and a tactical IMU sensor. This shows the positioning error bounds with just an IMU and it compares it to that of the full LiDAR sensor simulation. The full simulation using a baro sensor, the Doppler LiDAR sensor, and a navigation grade IMU.

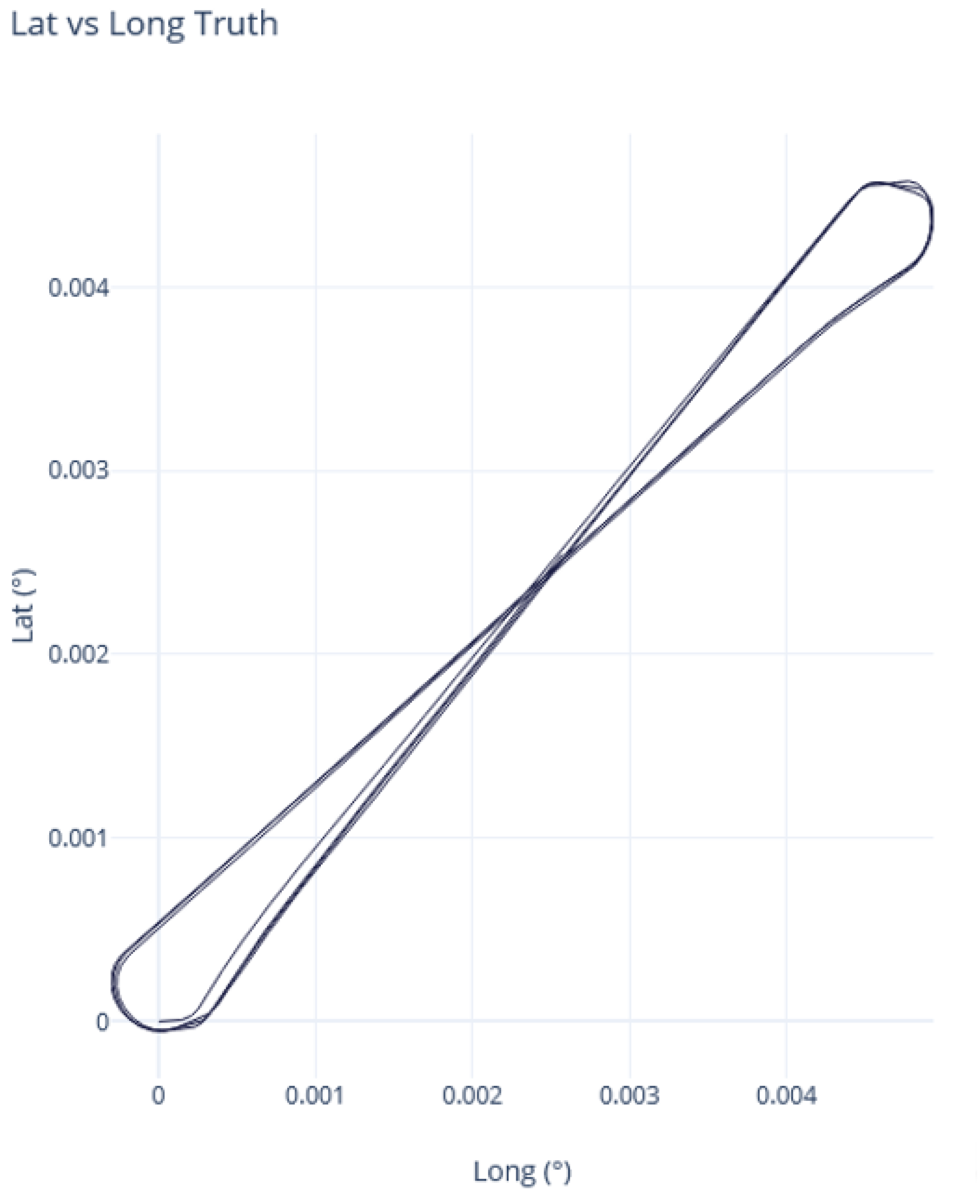
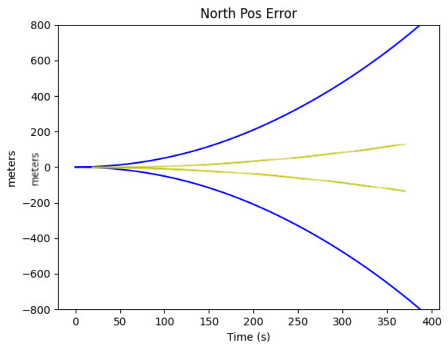


Figure 8: Flight path of truth data used in simulation.

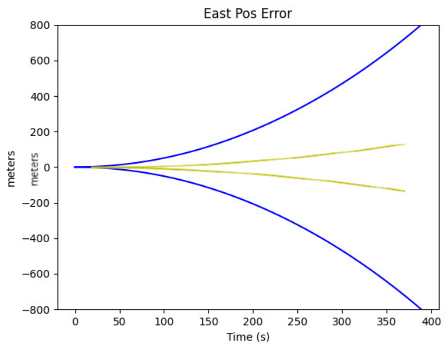
Table 2: Simulation initial conditions.

Doppler LiDAR Simulation Parameters	Data Value
IMU Grade	Navigation Grade HG9900 Model
Doppler LiDAR Sigma	3 cm/s
Doppler LiDAR Interval	1 Hz
Baro Sigma	1 m/s
Baro Interval	1 Hz
Pinson 15 Initial POS	0.0, 0.0, 0.0 m
Pinson 15 Initial VEL	.0003, .0003, .0003 m/s
Pinson 15 Initial Baro	0.0002, 0.0002, 0.0002
Pinson 15 Initial Accel Bias	0.0002, 0.0002, 0.0002
Pinson 15 Initial Gyro Bias	1.45e-08, 1.45e-08, 1.45e-08

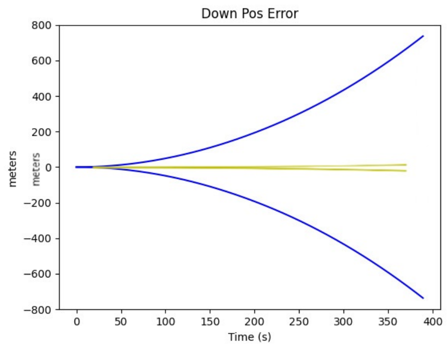
Table 3: Table to show simulation initial conditions.



(a) North



(b) East



(c) Down

Figure 9: NED Position Error with both a Navigation Grade IMU (Yellow) and Tactical Grade IMU (Blue) using covariance analysis. These plots were meant to verify that the basic covariance analysis was working as intended. This was confirmed since my navigation grade IMU out-preformed my tactical IMU.

East position error IMU only navigation scenario]East Position Error with both a Navigation Grade IMU (Yellow) and Tactical Grade IMU (Blue) using covariance analysis. These plots were meant to verify that the basic covariance analysis was working as intended. This was confirmed since my navigation grade IMU out-preformed my tactical IMU.

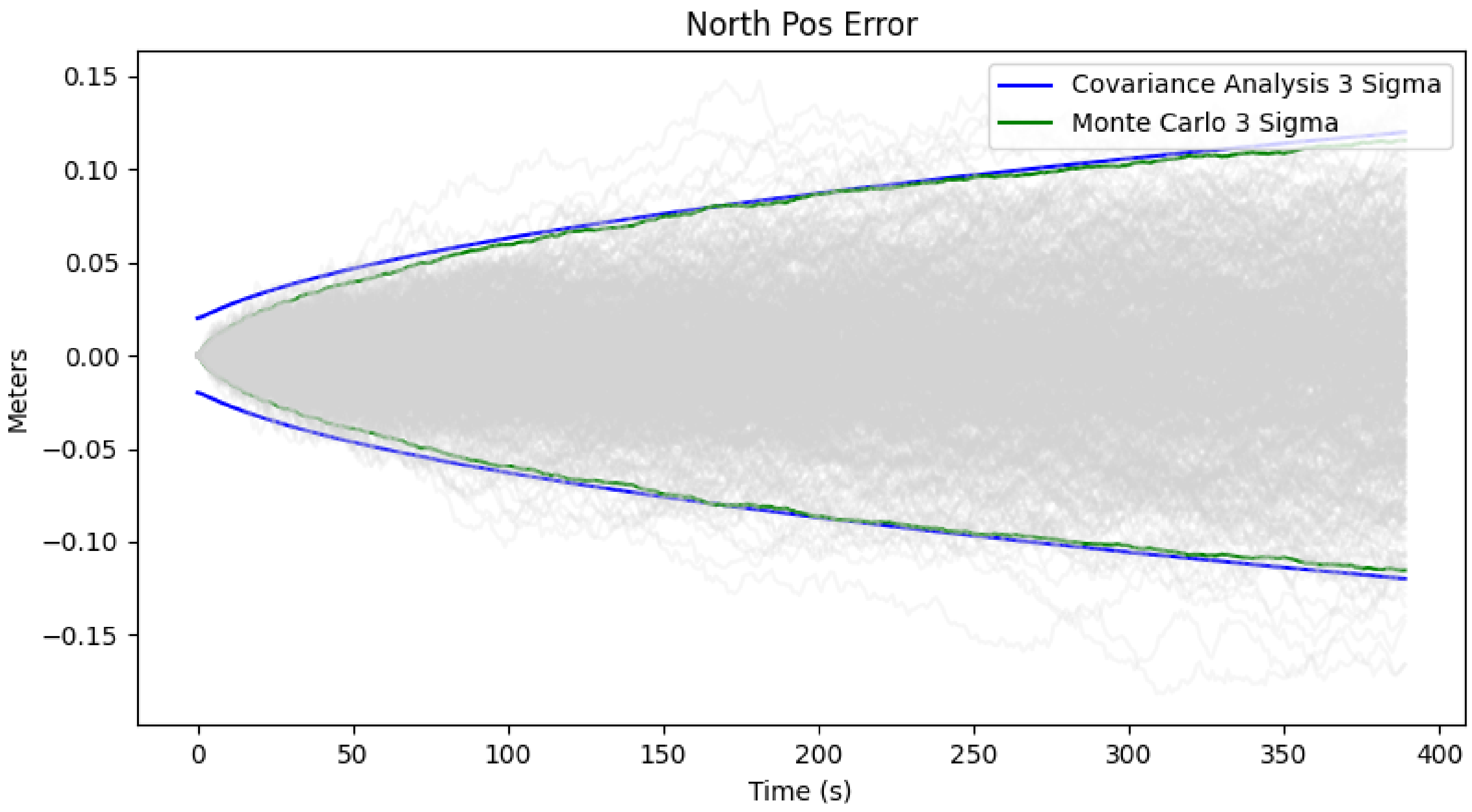


Figure 10: North Position Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows sub-meter accuracy using the Doppler LiDAR velocity sensor. This shows this sensor to be very promising in a navigation scenario.

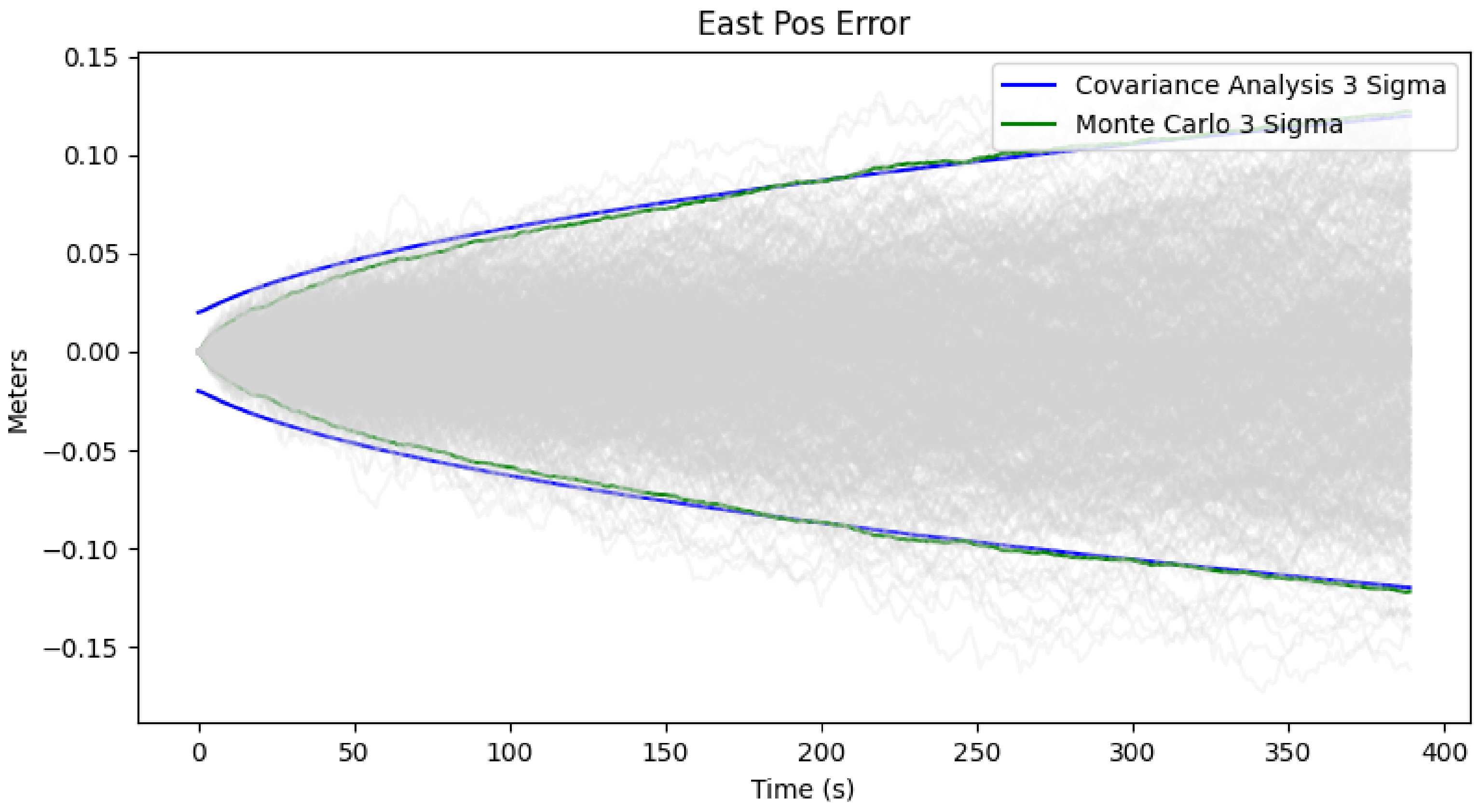


Figure 11: East Position Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows sub-meter accuracy using the Doppler LiDAR velocity sensor. This shows this sensor to be very promising in a navigation scenario.

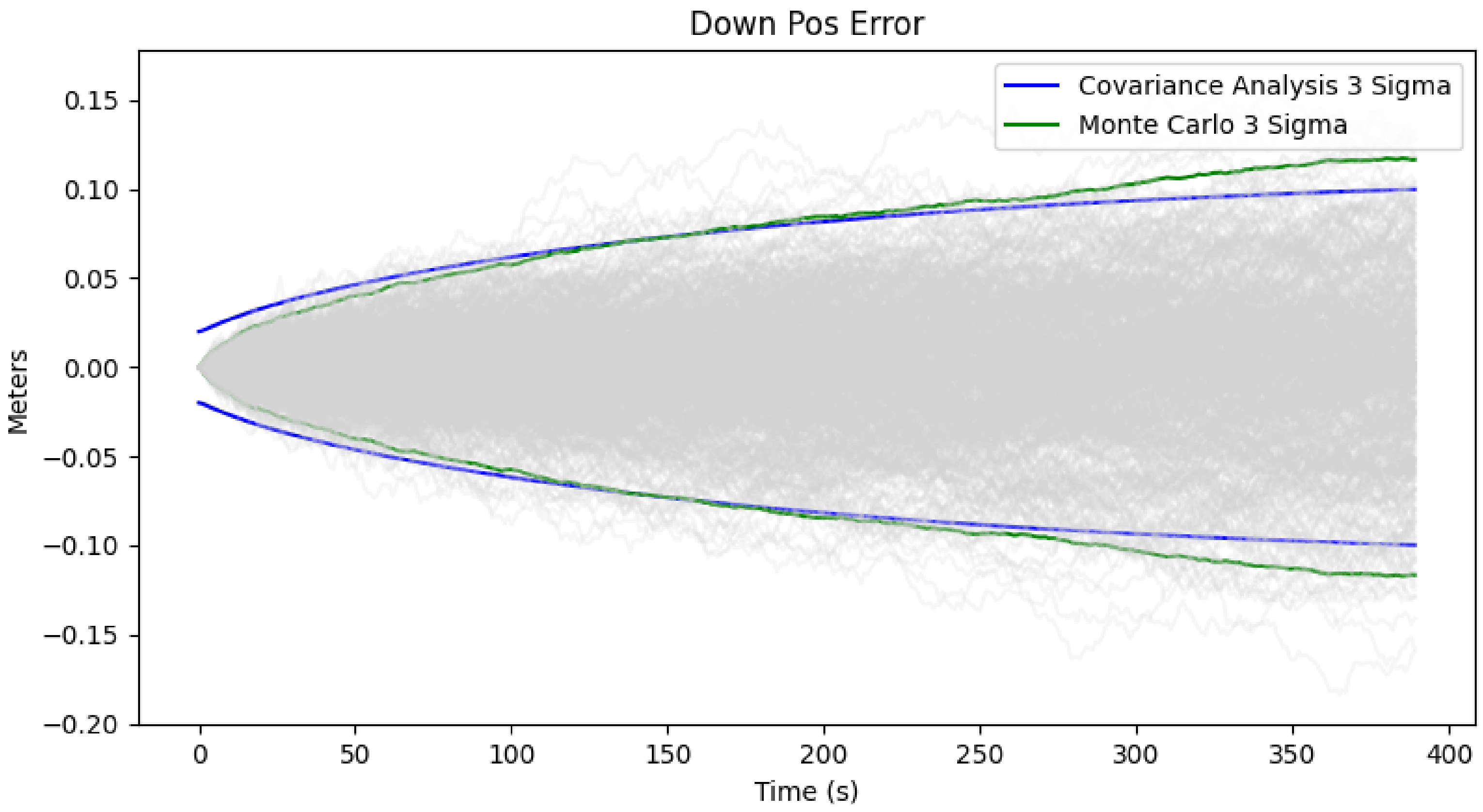


Figure 12: Down Position Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows sub-meter accuracy using the Doppler LiDAR velocity sensor. This shows this sensor to be very promising in a navigation scenario.

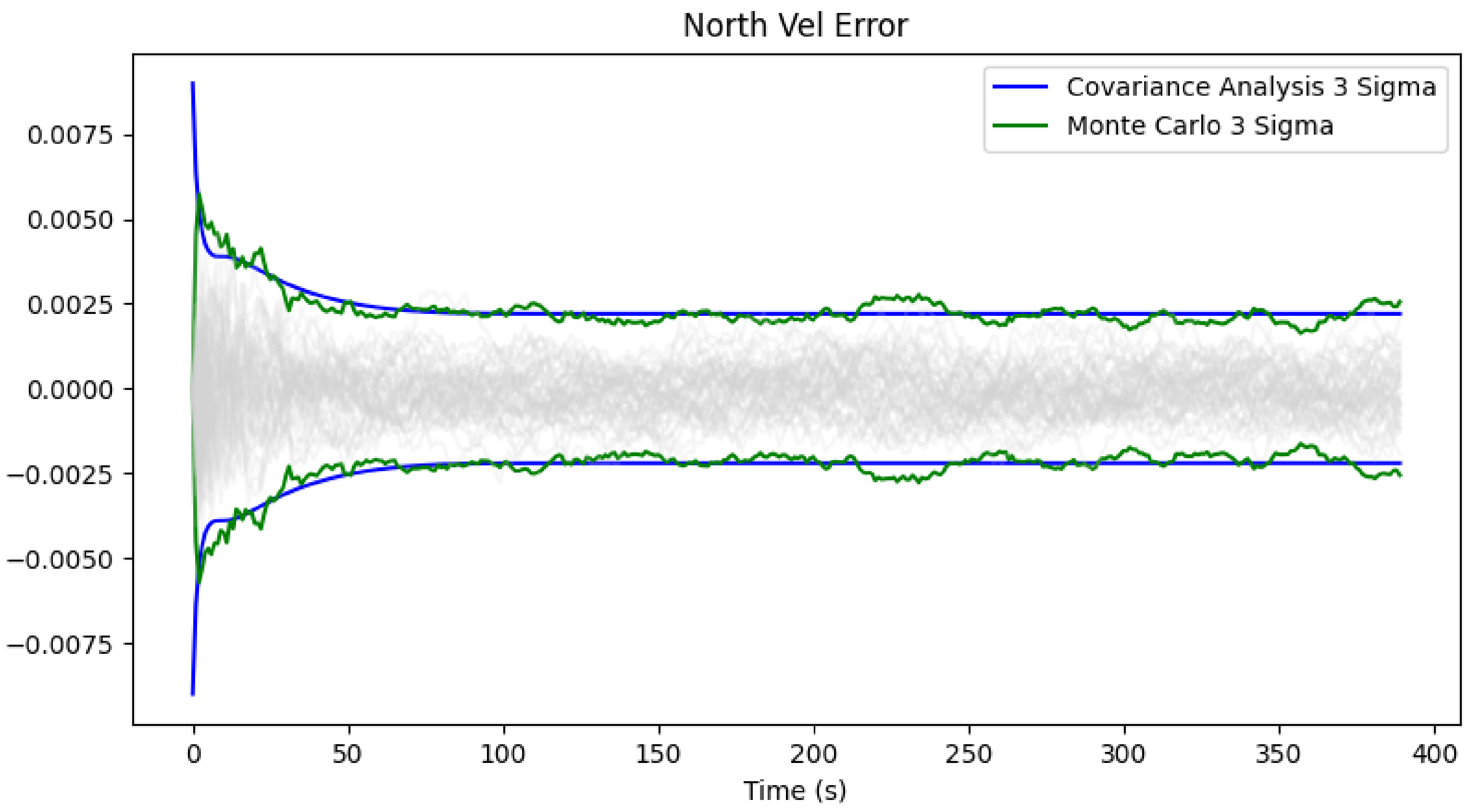


Figure 13: North Velocity Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows the 3 cm results in velocity since we are measuring it directly.

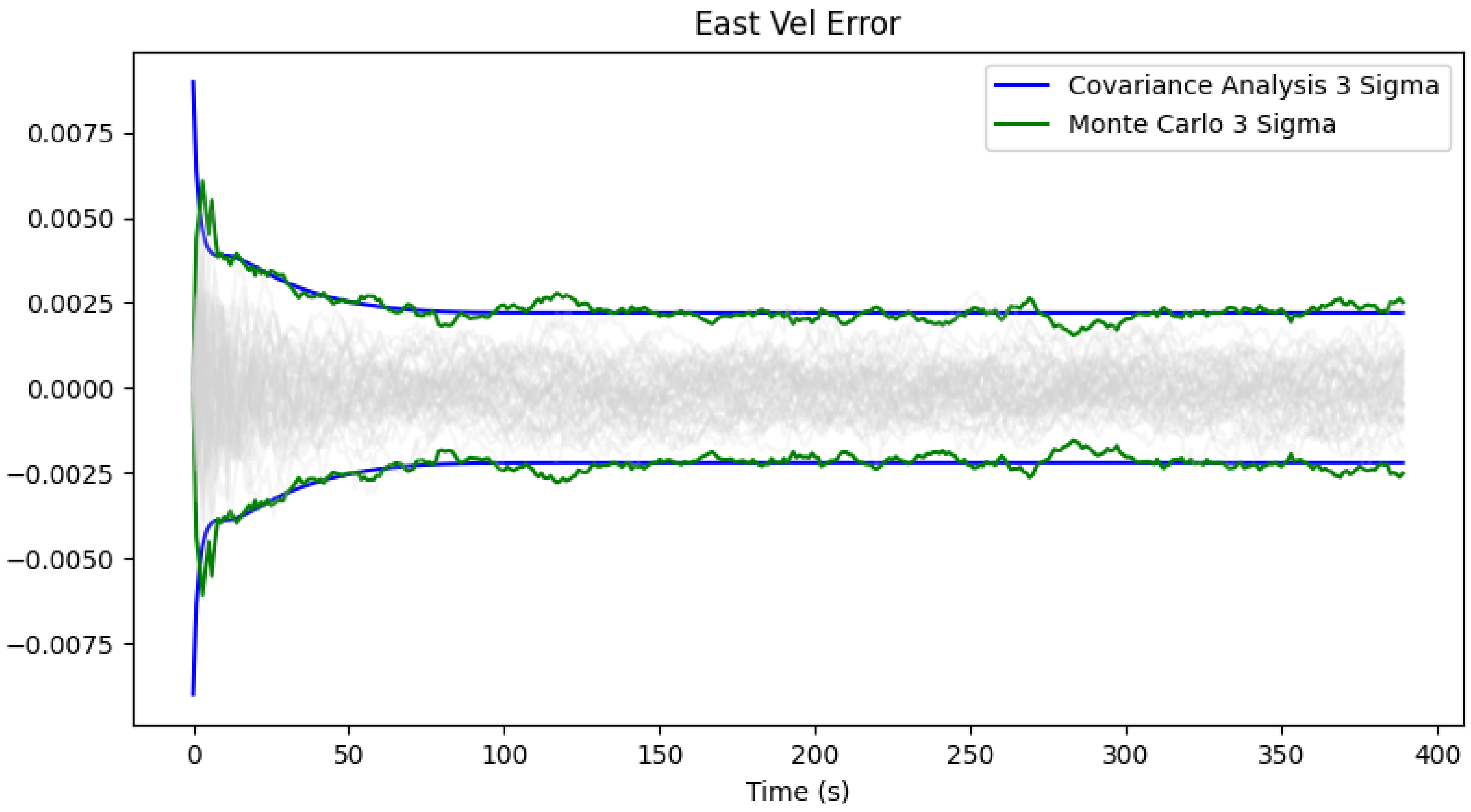


Figure 14: East Velocity Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows the 3 cm results in velocity since we are measuring it directly

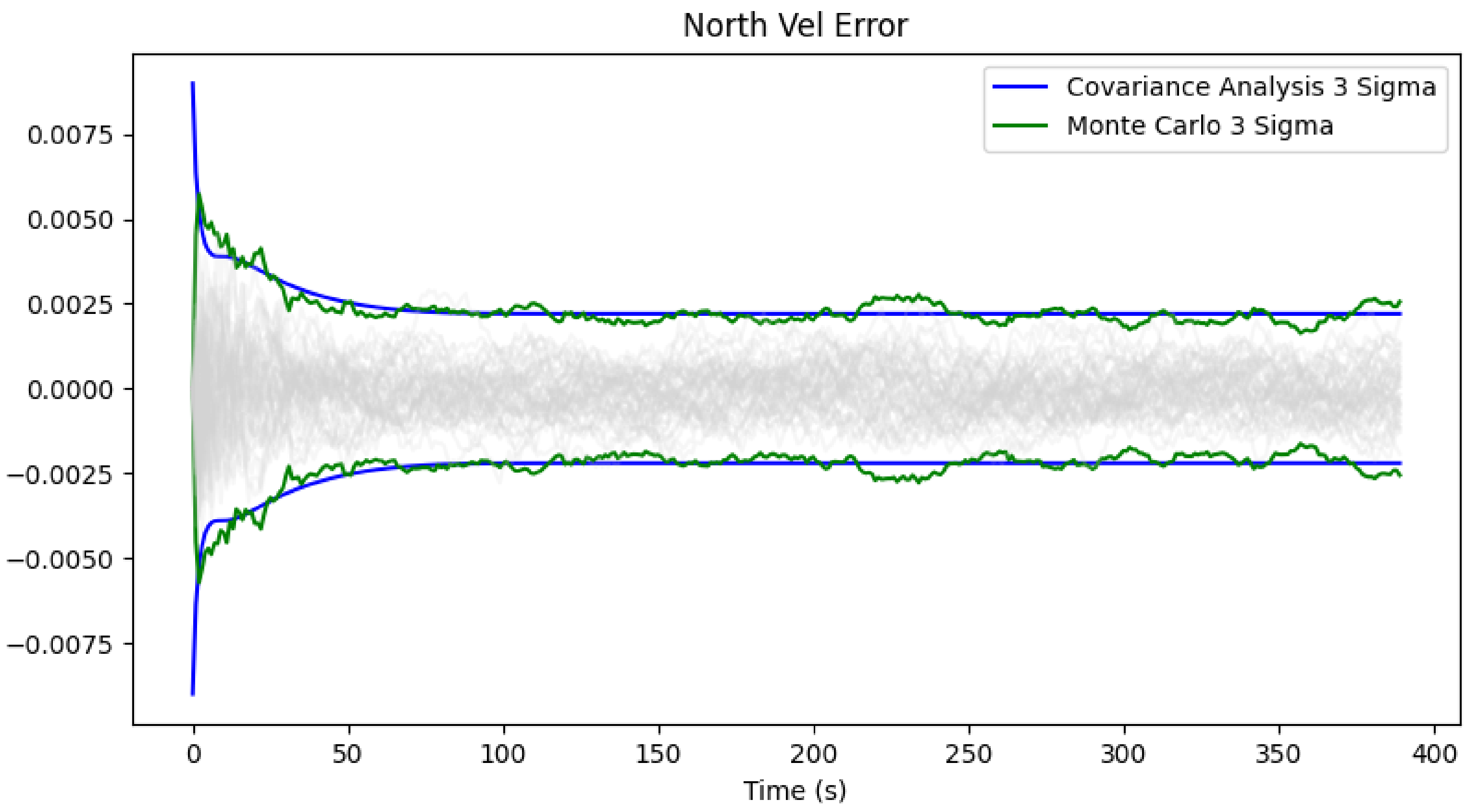


Figure 15: Down Velocity Error with both covariance analysis, Monte Carlo, and individual Monte Carlo runs in gray. This shows the 3 cm results in velocity since we are measuring it directly

3.5.1 Results

From the data collected we can see the Doppler LiDAR sensor's capabilities. Not only are the velocity measurements accurate, but the accuracy allows the navigation solution to keep a tight position solution for the period of time used. From the initial tactical IMU and navigation grade IMU analysis results, we can see even with the navigation grade IMU we are getting an error of approximately hundred and fifty meters in the north, east, and down direction after the six and a half minute analysis. The difference between the error of the navigation grade IMU only and the error results with the Doppler LiDAR sensor shows how significantly it can affect these measurements. With the Doppler LiDAR sensor having approximately 0.05 meters of error in the north, east, and down direction after the six and a half minute analysis had run and the IMU only having a large hundred and fifty meter error.

Further research will be done using a longer data set to see the length a position solution can be held with small error before it exponentially grows. This analysis shows that in our 6.5 minute flight positioning can be held to a high standard using the vendor Doppler LiDAR sensor. The results from this preliminary analysis lead us to interest in further research with this new sensor and gives researchers insight into the position/velocity accuracy this sensor can provide.

From our full analysis plots with the covariance analysis as well as the Monte Carlo, we can see that our 3 Sigma standard deviation for each lines up. This works as a sanity check to make sure our covariance analysis is working as expected and achieving the same results as a Monte Carlo analysis of the same data. This shows that the covariance analysis software is working as intended, and we can pursue further work using this software.

3.6 Conclusion

We have described a LiDAR sensor that with sufficient analysis may lead to a excellent position solution for a short duration. We gave a brief intro to covariance analysis as well as a description of the software developed and used for the analysis. We finished with the data we obtained from the covariance analysis and an analysis of the results. We hope that with longer truth solutions, we can learn more about the capabilities of this sensor and explore more sensor combinations and flight scenarios. Trade-space analysis like this LiDAR sensor analysis is vital to exploration into new sensors and forms of navigation. With a thoroughly detailed and easy to use software package, researchers will be able to conduct this type of research with ease and be able to take advantage of the low cost covariance analysis simulation brings in the realm of time and monetarily. With more sensors added to the software framework over time and an ever-growing amount of developers/users, the versatility of the software framework will only grow.

IV. Scholarly Article: Covariance Analysis Simulation Tool Demonstrated with Alt Nav Analysis

4.1 Introduction

The evaluation of different sensor pairings, error budgets, and different disturbances are vital analyses that need to be accomplished with alternative navigation. Executing tests in the real-world can be extremely expensive and time consuming. Creating simulations for these experiment parameters and running a simulated flight is much more practical and can provide similar, insightful data. In contrast to computationally expensive Monte Carlo analyses, Covariance Analysis can provide distribution information in one simulation run. Many papers utilize covariance analysis for different individual analysis problems such as closed loop navigation [16], space rendezvous [20], and lunar landers [2]. Currently, most covariance analysis research defines a specific problem to be researched and shows the results of said experiment using covariance analysis. This is great in helping the reader know how covariance analysis works as well as helping the reader know the pros and cons of the specific problem. This does not, however, give the reader the tools to run their own covariance analysis and set them up to be able to do their own research.

In this paper, we document the creation of a covariance analysis tool utilizing a new modular multi-source navigation sensor fusion library Navigation Toolkit[4]. The demonstration of the utility of the covariance analysis tool is done by analyzing the navigation performance of fusing a Magnetic Anomaly Navigation (MagNav) navigation solution [5] with a new Doppler Light Detection And Ranging (LiDAR) velocity sensor(CITE or Footnote). The covariance analysis of these two sensors helps to quantify the potential accuracy and practically in a full navigation solution. This paper was created for the reader to get an idea on how to use this new covariance analysis tool. Also, the reader will get a concept on how these two new navigation sensors can provide a GPS-alternative navigation solution and can be a game changer in the world of navigation.

Section 4.2 characterizes Monte Carlo analysis as well as covariance analysis and gives a brief introduction to trade-space analysis. Section 4.3 defines the software package used and describes how the software package is modified to allow the creation of covariance analysis simulations. It also describes the different ways users can operate the covariance analysis simulation tool. Section 4.4 defines the specific experiments conducted to prove the created covariance analysis simulation works as intended and shows the capability of the sensor. Finally, section 4.5 summarizes the contents of the effort.

4.2 Background

4.2.1 Trade-Space Analysis

The evaluation of different sensor pairings, error budgets, and different disturbances are vital analyses that need to be accomplished with alternative navigation. Executing these tests in the real-world, especially given the broadness of the experiments required, can be extremely expensive and time consuming. Creating simulations for these experiment parameters and running a simulation is much more practical and can provide the same data. A covariance analysis ran on a simulation with all parameters for the given sensors set prior to the run can give these required results quickly and efficiently.

4.2.2 Monte Carlo Analysis

Currently, Monte Carlo analysis is an extremely common tool for trade-space analysis and for good reason. Monte Carlo can work with nonlinear and non-Gaussian problems and provide complex statistical results. However, it requires hundreds, and often thousands, of simulated runs to obtain the required results. This leads to a considerable computational load and a lengthy run-time. Monte Carlo analysis is used in a wide scope of scientific fields and is a versatile tool.

In this specific case in a Guidance, Navigation, and Control (GNC) system, a Monte Carlo simulates the sensors with given specifications, computes the covariance and error states by generating a large amount of samples of each state, and then estimates the covariances.

In the following Figure (Fig. 16) you can see a flow diagram of a Monte Carlo simulation for navigation.

4.2.3 Covariance Analysis

Covariance analysis can provide similar statistical data as a Monte Carlo but in a single simulation run. This eliminates the computational burden that comes with the Monte Carlo analysis. Covariance analysis has a few extra requirements that are critical to keep in mind. Covariance analysis requires that the truth model be linearized. Therefore, if the situation induces many non-linearities, a Monte Carlo may be the only option. One must explicitly define the structure and uncertainties of the truth model and the design model for which the filter is based [6]. Also, all errors must be Gaussian distributed. Since all errors must be Gaussian and the equations linear, the covariance relationships are independent of the specific measurement time history [6]. This allows us to obtain the covariance, and thus the distribution information, of the simulation without the estimate. The following equations show how we arrive at the covariance without the estimate, the following equations assume a full Extended Kalman Filter which was defined above. Maybeck defines a full Monte Carlo analysis then defines the changes in the equations to arrive at covariance analysis [6]. $P_e(t)$ is

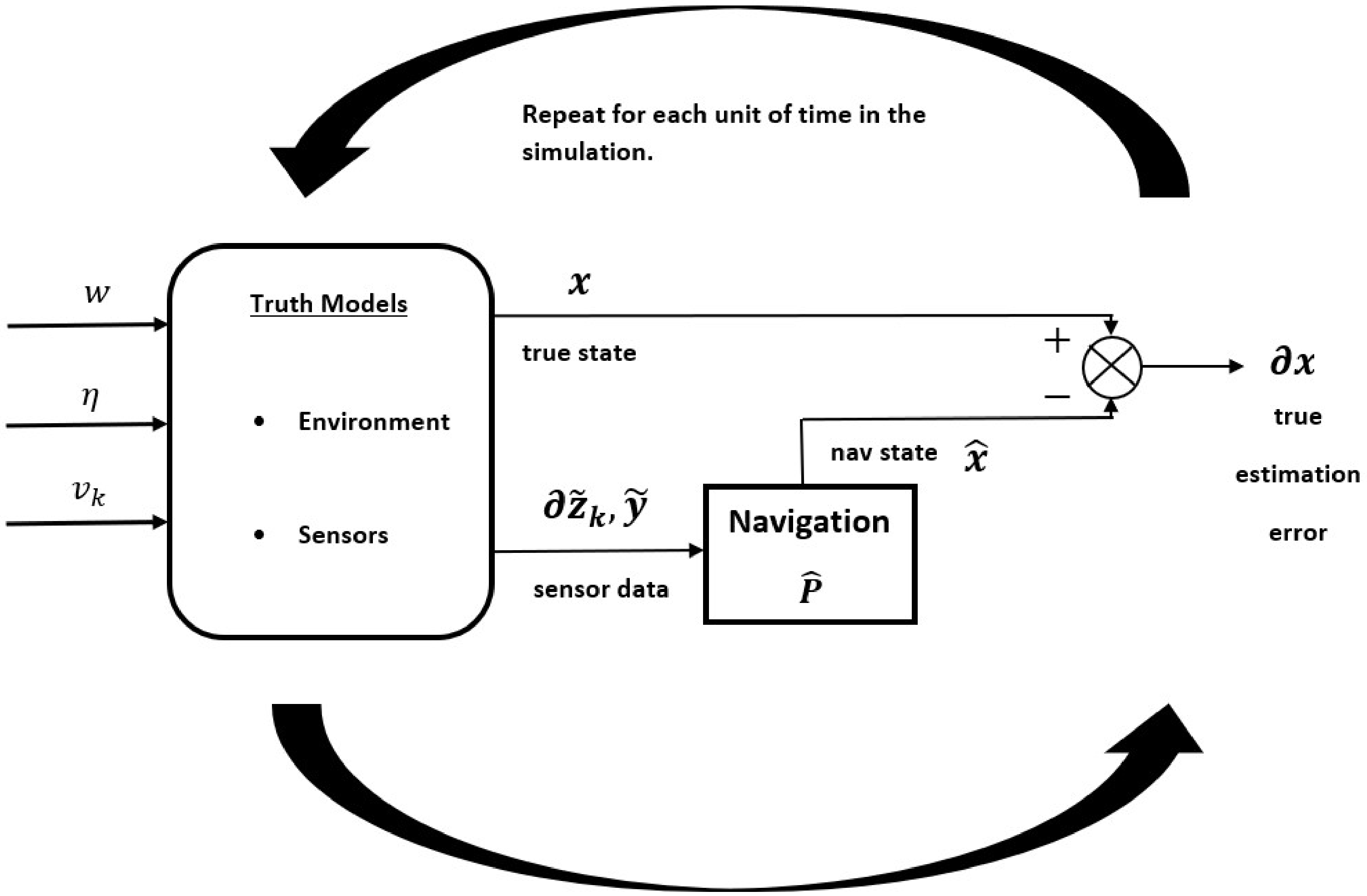


Figure 16: Monte Carlo simulation for a generalized GNC problem. This loop is ran for each time-step in the trajectory till one has completed the simulation. This represents a single estimate, with hundreds or thousands of these estimates being required. Then, the statistics of the errors in the system are the final outputs. The process noise w, continuous sensor noise η (e.g. the inertial measurement unit (IMU)), and discrete sensor noise v_k form the dynamics for the truth state x. The simulated sensor data is defined as $\delta \tilde{z_k}$, which are discrete measurements by sensors, and $\tilde{y}$, which are continuous IMU data. Finally, δx is the difference between the true state and the estimated state. The output value is δx. The mean value obtained from the Monte Carlo simulation is a statistical approximation of the true value. The statistical sample can be improved by increasing the total number of Monte Carlo runs.

the covariance of the true estimation errors committed by the given filter and $P_e(t)$ is the covariance of the augmented state. Since all processes are assumed to be zero mean, we can define $P_a(t)$ and $P_e(t)$ as

$$P_a(t) = E\{x_a(t)x_a^T(t)\}, \tag{45}$$

$$P_e(t) = E\{e_t(t)e_t^T(t)\}, \tag{46}$$

where e_t is the error committed by the filter and x_a being the augmented state vector at the given time. Propagating between samples is done by integrating

$$\dot{P}_a(t) = F_a(t)P_a(t) + P_a(t)F_a^T(t) + G_a(t)Q_t(t)G_a^T(t), \tag{47}$$

with a linear system with structure defined as defined as F_a and G_a. With uncertainty defined as Q_t The measurement update can be derived as

$$P_a(t_i^+) = A_a(t_i)P_a(t_i^-)A_a^T(t_i) + K_a(t_i)R_t(t_i)K_a^T(t_i), \tag{48}$$

where our initial covariance is defined as

$$P_a(t_0) = \begin{bmatrix} -P_{t0} & 0 \\ 0 & P_0 \end{bmatrix} \tag{49}$$

we then define the follow equations

$$A_a(t_i) = \begin{bmatrix} I & 0 \\ K(t_i)H_t(t_i) & [I - K(t_i)H(t_i)] \end{bmatrix}, K_a(t_i) = \begin{bmatrix} 0 \\ K(t_i) \end{bmatrix}, \tag{50}$$

A_a is required to update our covariance analysis equation and K_a being the Kalman gain of our estimate with zeros added on top.

Finally the desired error covariance can be obtained with

$$P_e(t) = C_a(t) P_a(t) C_a^T(t). \tag{51}$$

where

$$C_a(t) = \begin{bmatrix} -C_t(t) & C(t) \end{bmatrix} \tag{52}$$

. C is the critical states or quantities we care about. If we are only interested with a certain number of states in our system using C we can only return the ones that we are interested in. If we are interested in all of the states $C(t) = I$. We end up with the covariance of the state using the sensor covariance values and without having to calculate individual estimates. This saves a large amount of computation time.

4.2.4 Doppler LiDAR Velocity Sensor

A novel Doppler LiDAR sensor was originally developed by NASA and has been transferred to a company, Psionic, for further development and maturation. This sensor can make highly accurate, three-dimensional velocity measurements in the body frame of a vehicle. However, the practicability of such a sensor in an IMU-mechanized navigation solution is unknown. A covariance analysis is ran with a simulation flight across multiple trajectories as well as different IMU grades to understand the sensors accuracy and potential contributions to a navigation system. The company states the sensor can produce measurements with covariance around 3 cm/sec at a rate in the kilohertz.

4.2.5 Magnetic Navigation

Magnetic anomaly navigation is a Global Positioning System (GPS)-alternative navigation system that works by matching magnetometer measurements from a sensor to maps of the Earth's magnetic anomaly field [5]. Commonly called MagNav, this form of navigation is a nearly unjammable passive navigation system that works in any environment, day or night, as long as you have a map of the area. The magnetic field of Earth is the combination of many different sources. What the sensor measurements are actually measuring and matching to is denoted in Figure 19. Subtracting out the core field to get the projection of the crustal field (anomaly) on the core field.

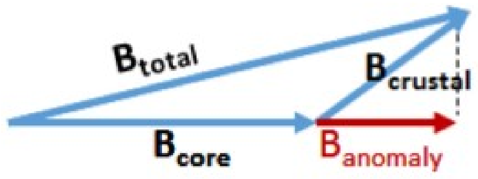


Figure 17: Magnetic Anomaly Definition [5]

MagNav is a well-documented form of alternative navigation that has been used in real-world flight tests and achieved eye-opening results [5, 8, 9, 10]. For example, 59 meter DRMS errors at 300 meters above ground and 111 meters DRMS errors at 1.5 kilometers above ground were achieved during a Test Pilot School (TPS) demonstration on an F-16 with sensors mounted in a RASCAL pod [8]. The main component that increases error in the MagNav sensor measurements is sensor/platform calibration. This is due to the induced magnetic field of the aircraft. The first goal is to place the MagNav sensor in a good position, e.g. as far away from sources of ferrous metal as possible. This is an error source we do not have to worry about since it will not be present in a simulation environment. Next, the magnetic sources in the aircraft can be removed by creating a calibration profile of the aircraft. This calibration will be explained more in depth when the full experiment parameters are defined.

Using covariance analysis with a large amount of test data and real magnetic anomaly maps, we can document the likely results of a navigation solution that includes a MagNav sensor. Conducting a covariance analysis will allow us to analyze a MagNav aided navigation solution and conduct different trade-space analysis techniques to continue to show the benefits of a MagNav aided navigation solution.

4.3 Software Description

A full description of Navigation Toolkit inside of Position, Navigation, and Timing Operating System (pntOS).

4.3.1 Navigation Toolkit

Navigation Toolkit is a Government-owned, modular Bayesian estimation software library designed to assist users in the creation of navigation filters. Nav Toolkit is being actively developed to build Viper plugins that are the government instantiation of the new pluggable pntOS software [3, 21]. All of this software hails from open-source Bayesian estimation software developed originally at the ANT Center called Scorpion[4]. This software setup allows researchers to easily modify sensor values, add and remove sensor suites, as well as swap filters with minimal changes to code.

Navigation Toolkit works by way of fusion engines/strategy, state blocks and measurement processors; a full decomposition of this can be seen in Figure 19. The fusion engine/strategy sets up the users filter of choice. As of now the options out of the box are an unscented Kalman Filter, an Extended Kalman Filter, and Rao-Blackwellized Particle Filter. For the example in this paper, we will use an extended Kalman Filter (EKF) since this works with the covariance analysis requirements well.

Stateblocks State blocks are a set of N states, these state blocks house the info required to propagate the states forward in time. For the analysis, the stateblock to use is an out of the box Pinson15Ned state block. This pinson15 stateblock update as the filter runs and what happens inside of them will be described in detail in their own respective sections. The measurement processor houses a model of the sensor you have called and instructions on how the measurement will be included in your used state blocks. A decomposition of a stateblock can be seen in Figure 18.

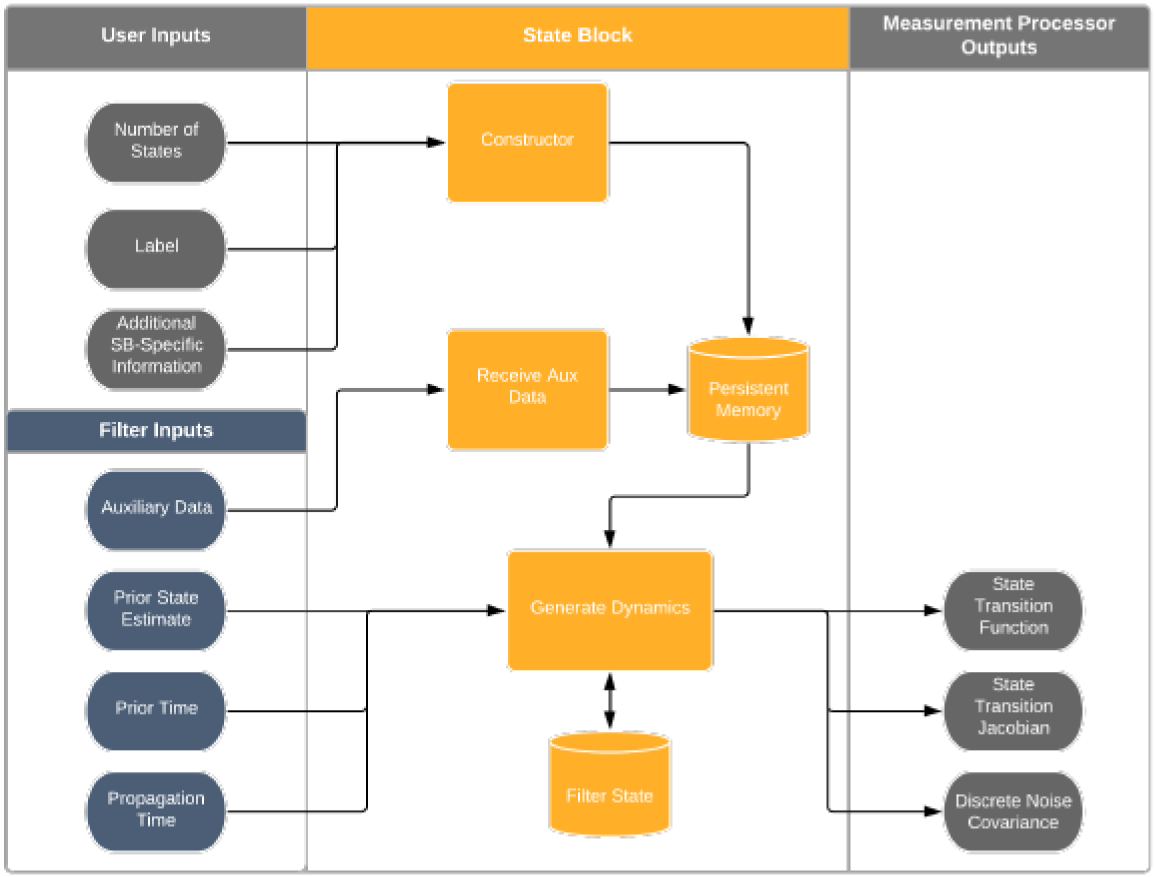


Figure 18: Diagram of the Navigation Toolkit Framework [4]. This diagram shows a generic stateblock. Displays how the user inputs the amount of states and gives the stateblock a label. The filter inputs estimates and the stateblock generates dynamics. Depending on the measurement processors, they will take the info generated from the stateblock and update.

Measurement Processors Given a sensor measurement, the measurement processor provides the fusion engine with a model to update the states. Given a set of observations that contain information about the state estimates, the measurement processor produces the model which relate the two.

Fusion Engines Finally the Fusion engine is what is called and puts everything together, the stateblock, measurement-processor as well as the raw measurements from the included sensors and produces the full estimates of all the states that are required by the stateblocks. The estimates are created for the past, present, and future of the states. It works by the measurement processor passing the engine the measurements then the models from the stateblocks and measurement processor go to the fusion strategy.

The ease of adding, editing, and deleting different sensors, using different filters, and utilizing different models lets us analyze the new Doppler LiDAR sensor to an extent unreachable before. pntOS exposes a framework that researchers can easily write their own models or filters in and have access to other researchers created filters/models. The creation of Viper was to allow researchers to work on navigation simulations or build sensors in a common place where other researchers can pull what they have done and add it to their work without having to reinvent the wheel.

One more item is needed for to propagate the simulation in a Covariance Analysis. We must have a nominal trajectory with time, about which to compute the covariance. This must be supplied, since we do not propagate a mean solution. In addition, we require the truth values for all of the stateblocks about this nominal trajectory. For example, for the Pinson-15, we require the true errors in position, rotation, velocity, and Inertial Navigation System (INS) biases with time. Trajectory information must be specified as position, velocity and attitude with time. These should be 6 degree of freedom trajectories to receive the highest performance.

The tool described below uses an out-of-the-box Pinson15Ned state block. The stateblock updates as the filter runs and what they do will be described in its own section below. Also, the measurement processors we are using will be explained in depth below.

4.3.2 Pinson15 StateBlock

A pinson15 is a set of 15 state blocks bundled together that model the errors of an INS in a moving North East Down (navigation) frame. A Pinson15 requires an initial value for the 15 states used and we set this up with the initial sigma values of the given sensors as well as the initial bias in the gyro and accelerometer. A Pinson15 model can be described as the following [15]

$$X = \begin{bmatrix} \delta p_n \ \delta p_e \ \delta p_{vert} \ \delta v_n \ \delta v_e \ \delta v_d \ \epsilon_n \ \epsilon_e \ \epsilon_d \ b_{a_x} \ b_{a_y} \ b_{a_z} \ b_{g_x} \ b_{g_y} \ b_{g_z} \end{bmatrix}^T \tag{53}$$

where δp is n-e-vertical position errors, δv represents NED velocity errors, η are the tilt errors about the NED axis, b_a x,y,z are accelerometer time-correlated biases, and b_g x,y,z gyro time-correlated biases. The linearized dynamics model can be expressed as

$$\begin{bmatrix} \dot{\delta p} \\ \dot{\delta v} \\ \dot{\delta \epsilon} \\ \dot{b_a} \\ \dot{b_g} \end{bmatrix} = \begin{bmatrix} F_{pp} & F_{pv} & F_{p\epsilon} & 0_{3\times3} & 0_{3\times3} \\ F_{vp} & F_{vv} & F_{v\epsilon} & C_s^n & 0_{3\times3} \\ F_{\epsilon p} & F_{\epsilon v} & F_{\epsilon\epsilon} & 0_{3\times3} & C_s^n \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{aa} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & F_{gg} \end{bmatrix} \begin{bmatrix} \delta p \\ \delta v \\ \delta \epsilon \\ b_a \\ b_g \end{bmatrix} \tag{54}$$

Before we define the submatrices we will define the variables used inside of them with the following table where the sub-matrices are defined as the following

$$F_{pp} = \begin{bmatrix} 0 & 0 & \frac{-v_n}{R_e^2} \\ \frac{v_e tanL}{R_e cosL} & 0 & \frac{v_e}{R_e^2 cosL} \\ 0 & 0 & 0 \end{bmatrix} \tag{55}$$

Table 4: Variable Definitions for Pinson 15 Error Model

Variable	Description
L	Latitude
v_n	North Velocity
v_e	East Velocity
v_d	Down Velocity
R_e	Earth Radius
f_n	Measured Specific Force North Direction
f_e	Measured Specific Force East Direction
f_d	Measured Specific Force Down Direction
Ω	Earth Rotation Rate

$$F_{pv} = \begin{bmatrix} \frac{1}{R_e} & 0 & 0 \\ 0 & \frac{1}{R_e cosL} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{56}$$

$$F_{p\epsilon} = 0_{3\times 3} \tag{57}$$

$$F_{pv} = \begin{bmatrix} v_e(2 * \Omega cosL \frac{v_e}{R_e cos^2 L}) & 0 & \frac{v_e^2 tanL - v_n * v_d}{R_e^2} \\ 2\Omega(v_n cosL - v_d sinL) + \frac{v_n v_e}{R_e cos^2 L} & 0 & \frac{-v_e(v_n tanL + v_d)}{R_e^2} \\ 2\Omega v_e sinL & 0 & \frac{v_n^2 + v_e^2}{R_e^2} \end{bmatrix} \tag{58}$$

$$F_{vv} = \begin{bmatrix} \frac{v_d}{R_e} & -2(\Omega sinL + \frac{v_e}{R_e cos^2 L}) & \frac{v_n}{R_e} \\ 2\Omega sinL + \frac{v_e tanL}{R_e} & \frac{v_n tanL + v_d}{R_e} & 2\Omega cosL + \frac{v_e}{R_e} \\ \frac{-2v_n}{R_e} & -2(\Omega cosL + \frac{v_e}{R_e}) & 0 \end{bmatrix} \tag{59}$$

$$F_{v\epsilon} = \begin{bmatrix} 0 & -f_d & f_e \\ f_d & 0 & -f_n \\ -f_e & f_n & 0 \end{bmatrix} \tag{60}$$

$$F_{\epsilon p} = \begin{bmatrix} -\Omega sinL & 0 & \frac{-v_e}{R_e^2} \\ 0 & 0 & \frac{v_n}{R_e^2} \\ -\Omega cosL - \frac{v_e}{R_e cos^2 L} & 0 & \frac{v_e tanL}{R_e^2} \end{bmatrix} \tag{61}$$

$$F_{\epsilon v} = \begin{bmatrix} 0 & \frac{1}{R_e} & 0 \\ -\frac{1}{R_e} & 0 & 0 \\ 0 & \frac{tanL}{R_e} & 0 \end{bmatrix} \tag{62}$$

$$F_{\epsilon\epsilon} = \begin{bmatrix} 0 & -\Omega sinL + \frac{v_e}{R_e tanL} & \frac{v_n}{R_e} \\ \Omega sinL + \frac{v_e tanL}{R_e} & 0 & \Omega cosL + \frac{v_e}{R_e} \\ \frac{-v_n}{R_e} & -\Omega cosL - \frac{v_e}{R_e} & 0 \end{bmatrix} \tag{63}$$

where C_s^n is a direction cosine matrix, which takes a vector expressed in the body frame and expresses that vector in the navigation frame. F_{aa} describes the decaying exponential terms of the first order Gauss-Markov process given by

$$F_{aa} = \begin{bmatrix} \frac{1}{\tau_a} & 0 & 0 \\ 0 & \frac{1}{\tau_a} & 0 \\ 0 & 0 & \frac{1}{\tau_a} \end{bmatrix} \tag{64}$$

where τ_g is a time constant of the accelerometer time-correlated biases. F_{gg} is the same, but for the time constant of the gyro time correlated biases given as

$$F_{gg} = \begin{bmatrix} \frac{1}{\tau_g} & 0 & 0 \\ 0 & \frac{1}{\tau_g} & 0 \\ 0 & 0 & \frac{1}{\tau_g} \end{bmatrix}. \tag{65}$$

4.3.3 MagNav Measurement Processor

MagNav measurement processor utilizes the magnetic anomaly map provided in the trajectory to create estimates sensor measurements. By taking the trajectory from the magnetic anomaly map and using the input sigma value for the magnetometer we create simulated measurements for the navigation solution to use. A full derivation of how the magnetic anomaly maps are used to create position estimates can be found in papers by Canciani, McNeil, and Bonifaz [9, 10, 5, 8].

4.3.4 Psionic Measurement Processor

The Doppler LiDAR measurement processor is only a basic velocity sensor inside of navigation toolkit. By changing the velocity sensor sigma and update rate to that of the Doppler LiDAR sensor, we can simulate it in the covariance analysis. The company states the sensor can produce measurements with covariance around 3 cm/sec at a rate in the kilohertz.

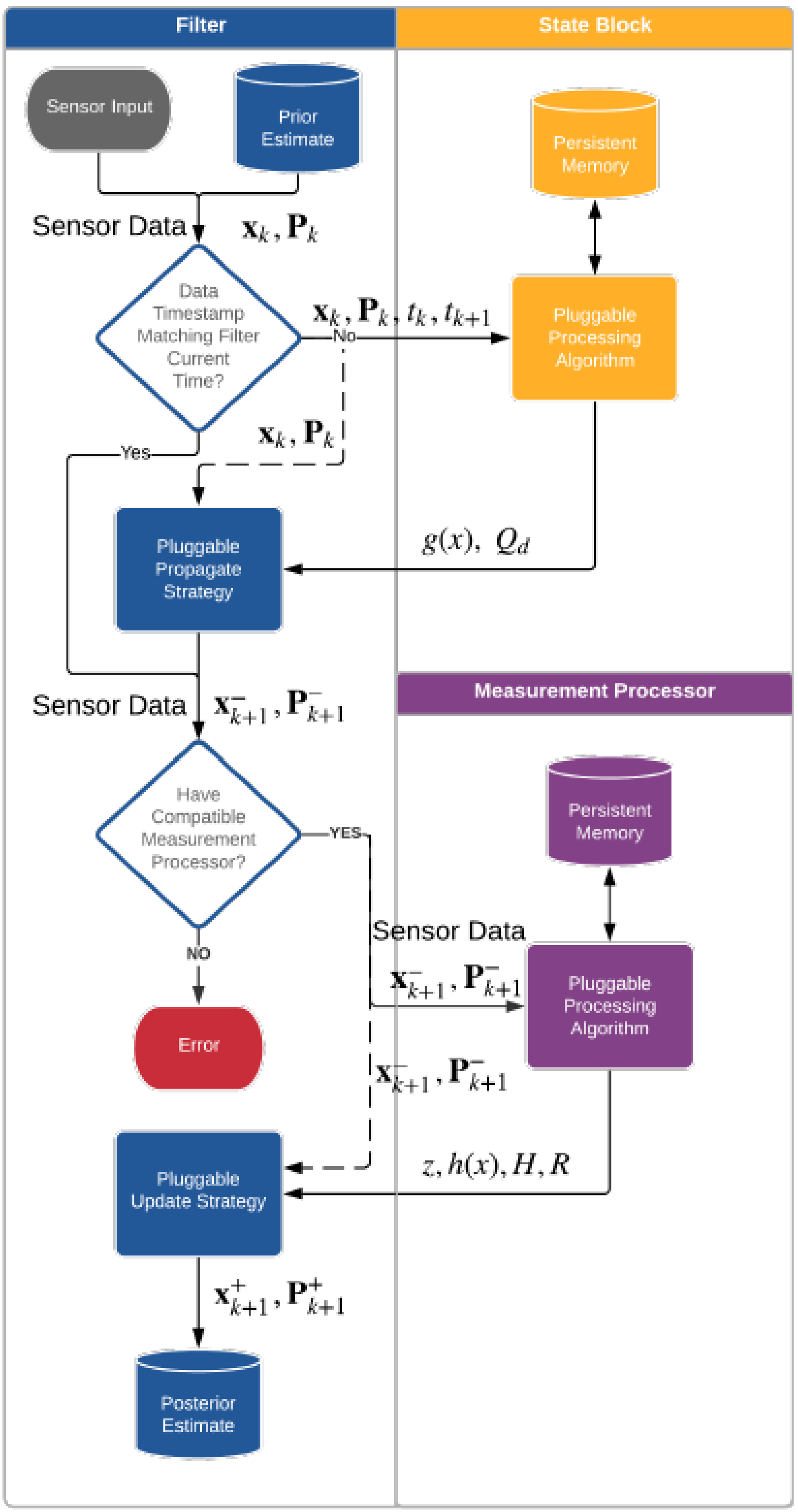
4.3.5 Covariance Analysis Simulation Tool

The covariance analysis tool is a simulation that utilizes pntOS to simulate navigation scenarios. The user will input the required info and it will output plots of error state covariance using the pinson15 error model. pntOS gives the ability to utilize premade filters like the EKF and models like the pinson15. pntOS will allow for the use of many different models and filters if the user wishes to use one they made themselves or any of the pre-made ones in navigation toolkit.

The simulation in Navigation Toolkit takes in a zip of the flight's truth. The filter runs with an EKF, updates the three state block models, and creates simulated sensor measurements for all time intervals. The covariance analysis as described above has a few requirements that are easily met with the EKF. The errors must be Gaussian and the navigation equations must be linearized. Since we are using a Pinson15 error model, we can assume the truth is zero for all time.

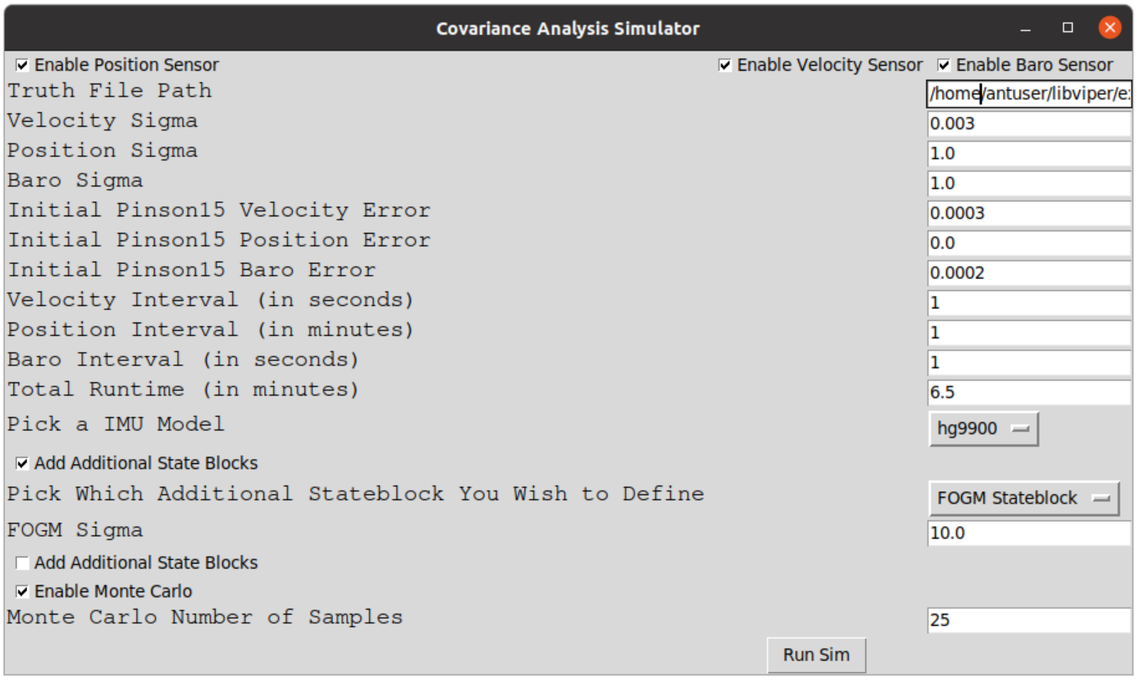
This simulation tool will be built on over time to add more and more different sensors, stateblocks, and ease of usability for the User Interface (UI) version of the tool. Currently, there exists a terminal tool, built in C++, that is wrapped in python bindings and a UI version that allows users to quickly use the tool without any technical training with the tool itself. The UI version is more limited as one must use the stateblocks, sensors, and models already added by selecting different check boxes or drop downs. These can be seen in a UI depicted in Figure 20. This helps users that are not as accustomed to the tool or users who simply want to quickly run a simulation and get results. A user can easily run the covariance simulation program, pick their IMU/sensors, pick the different stateblocks they wish to run, give the path to their trajectory Comma-Separated Values (CSV), and give all the relevant sigmas/intervals for the sensors.

If users want to create their own stateblocks, their own custom IMUs, or further customize the tool, the terminal version is what they would use. By using one the many examples navigation toolkit offers as a template. Users can easily set the values of their own stateblocks to add custom models or can call on the custom IMU and set their own sigma values. The "straight_flight_example" view from a IDE can be seen in the appendix A. This shows where users can add their own custom IMUs, edit sigma values, enable and disable sensors, set where the trajectory for the simulation is coming from and edit the interval at which sensors are updated.



Filter
State Block
Sensor Input
Prior Estimate
Sensor Data
$\mathbf{x}_k, \mathbf{P}_k$
Data Timestamp Matching Filter Current Time?
$\mathbf{x}_k, \mathbf{P}_k, t_k, t_{k+1}$
No
Persistent Memory
Pluggable Processing Algorithm
Yes
$\mathbf{x}_k, \mathbf{P}_k$
Pluggable Propagate Strategy
$g(x),\ Q_d$
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Measurement Processor
Have Compatible Measurement Processor?
YES
Persistent Memory
Sensor Data
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
Pluggable Processing Algorithm
NO
Error
$\mathbf{x}^-_{k+1}, \mathbf{P}^-_{k+1}$
$z, h(x), H, R$
Pluggable Update Strategy
$\mathbf{x}^+_{k+1}, \mathbf{P}^+_{k+1}$
Posterior Estimate

Figure 19: Diagram of the Navigation Toolkit Framework [4]. This diagram shows how the framework of pntOS works and how all 3 main components (filter, state-block, and measurement processor) work together to create a navigation simulation environment.



Covariance Analysis Simulator

[✓] Enable Position Sensor [✓] Enable Velocity Sensor [✓] Enable Baro Sensor

Truth File Path	/home/antuser/libviper/e
Velocity Sigma	0.003
Position Sigma	1.0
Baro Sigma	1.0
Initial Pinson15 Velocity Error	0.0003
Initial Pinson15 Position Error	0.0
Initial Pinson15 Baro Error	0.0002
Velocity Interval (in seconds)	1
Position Interval (in minutes)	1
Baro Interval (in seconds)	1
Total Runtime (in minutes)	6.5
Pick a IMU Model	hg9900

[✓] Add Additional State Blocks

Pick Which Additional Stateblock You Wish to Define	FOGM Stateblock
FOGM Sigma	10.0

[ ] Add Additional State Blocks

[✓] Enable Monte Carlo

Monte Carlo Number of Samples	25

Figure 20: Example of UI ran example. Users can use this to quickly input values for sensor sigmas, truth path, and even run a Monte Carlo on top of their covariance analysis if they so choose. This allows researchers to quickly run covariance analysis without learning pntOS.

4.4 Experiment

A nominal trajectory of an airplane flying over the Ottawa area is used for the analysis. A visual of this trajectory can be seen in Figure 21. A large, highly accurate magnetic anomaly map exists under the region covered by the trajectory. The base set of sensors include a GPS, a HG9900 IMU, and an altimeter. These sensors will be used in all simulations and the sigmas of these sensors will be defined in Table 5. These quality sensors contribute well in testing the newly created covariance analysis simulation tool and in researching the combinations of MagNav (using a magnetometer) and the Doppler LiDAR velocity sensors. We have simulations that will include both the magnetometer and Doppler LiDAR sensor, ones with a magnetometer and without the Doppler LiDAR sensor, and finally a navigation solution with the base set of sensors without the MagNav and Doppler LiDAR sensor. All simulations will start with 50 minutes of GPS data, then a GPS outage will occur for the remainder of the simulation.

The first simulation group (Figures 22 23 24) is two scenarios with 3500 seconds of the simulation including GPS, then a GPS outage for the rest of the run. The next figures are of the Ottawa Flight Simulation including a Doppler LiDAR velocity sensor as well as a MagNav magnetometer solution.

Table 5: Simulation initial conditions.

Simulation Parameters	Data Value
IMU Grade	Navigation Grade HG9900 Model
Doppler LiDAR Sigma	3 cm/s
Doppler LiDAR Interval	1 Hz
Baro Sigma	1 m/s
Baro Interval	1 Hz
Pinson 15 Initial POS	1.0, 1.0, 1.0 m
Pinson 15 Initial VEL	.0003, .0003, .0003 m/s
Pinson 15 Initial Baro	0.0002, 0.0002, 0.0002
Pinson 15 Initial Accel Bias	0.0002, 0.0002, 0.0002
Pinson 15 Initial Gyro Bias	1.45e-08, 1.45e-08, 1.45e-08
GPS Interval	1 Hz
GPS Sigma	10 m
Magnetometer Sigma	10 nanotesla

From these figures we can see that with this navigation solution and a loss of GPS the navigation solution can keep sub 5 meter positioning accuracy for the entire duration of the flight. This is a significant difference than if the velocity sensor is not included which will be presented in the next set of plots (Figure 25 26 27) . The next plots will be a basic navigation solution without the Doppler LiDAR sensor and without the magnetometer (Figure 28 29 32) . The error in position can be seen to increase 10 times from the plots that included only MagNav.

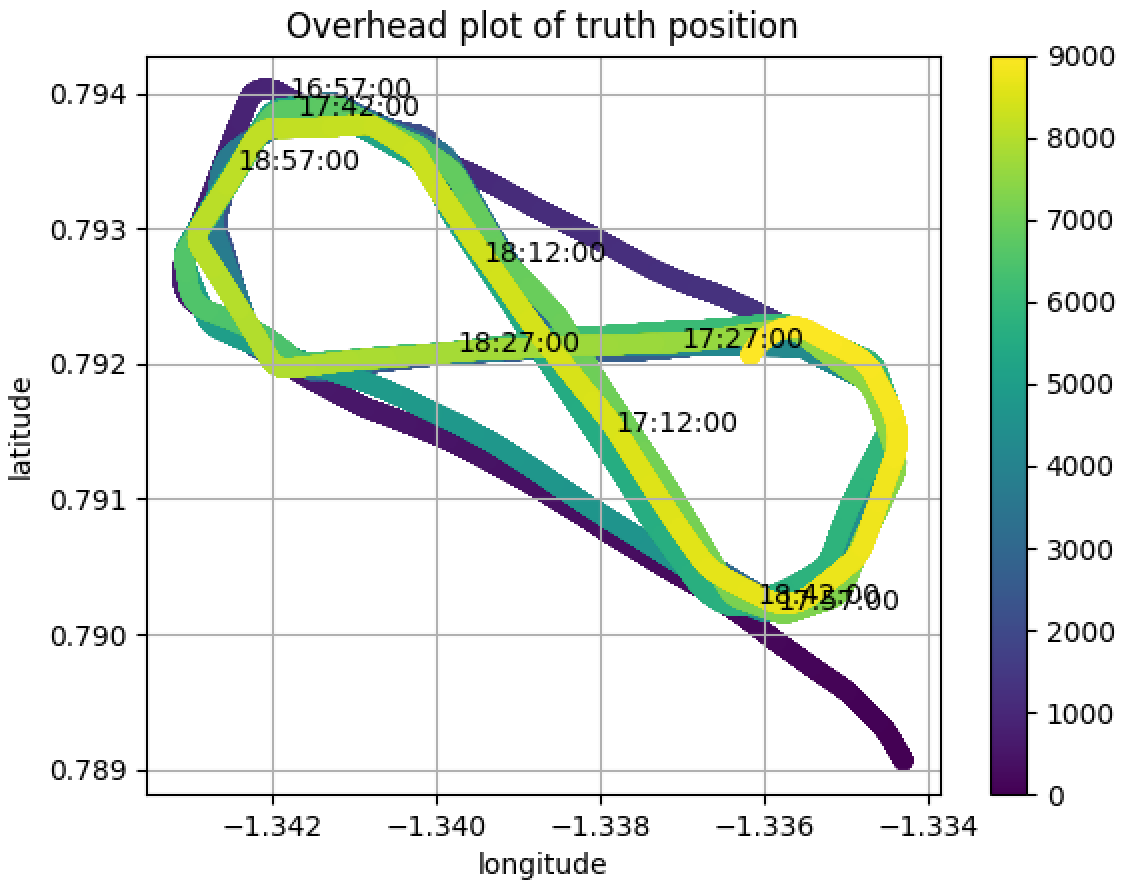


Figure 21: Plot of Ottawa flight truth position data (latitude vs longitude)

Covariance analysis can conduct error budgets to determine individual contributions to total error by different states[6]. Since the simulation uses a Pinson15 error

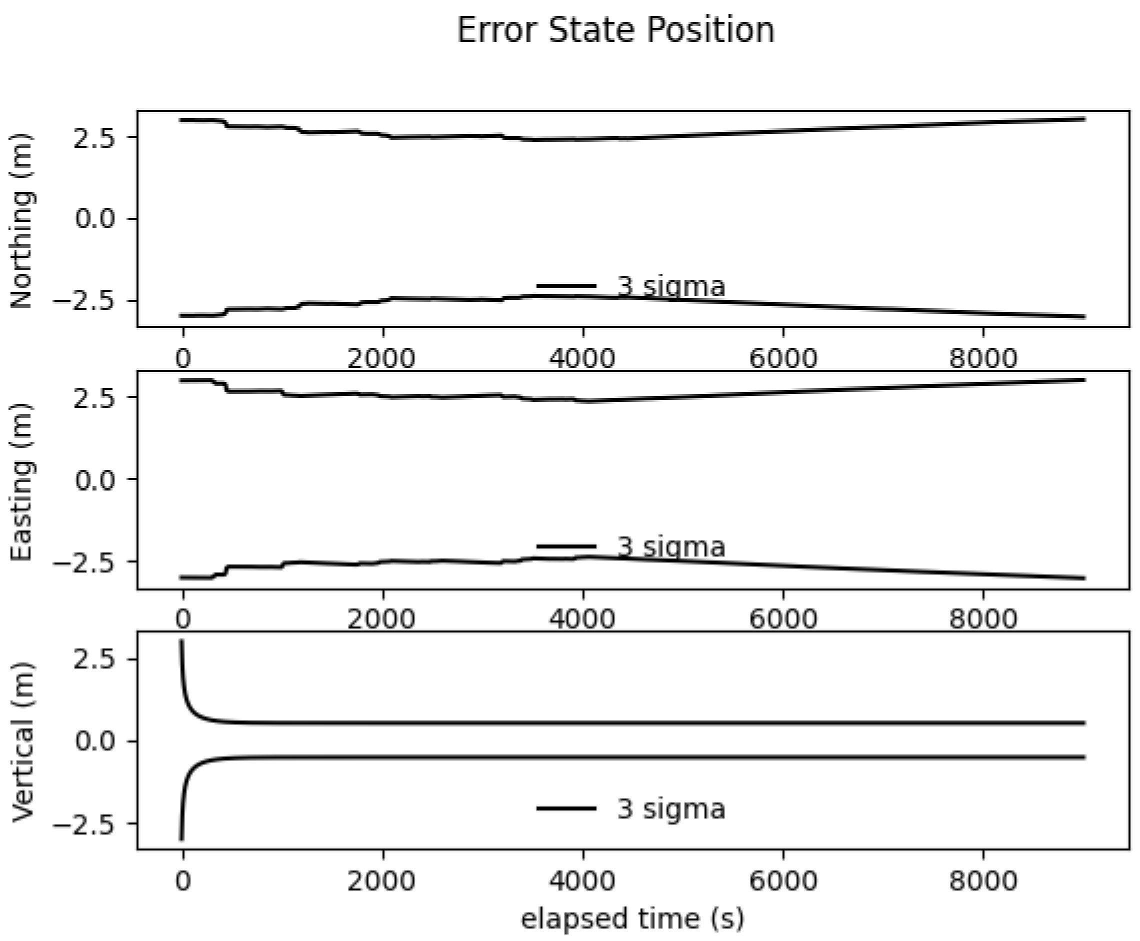


Figure 22: The position error state covariances are shown for the full simulation that includes the INS trajectory, velocity measurement update and MagNav measurement updates. Note that the 3-σ positions never exceed 6 meters in North, East or Down. These are remarkable results that beg to be investigated using hardware flight tests.

model, one can turn off error sources in position, velocity, and attitude. Then, enable them one at a time to determine which sources are producing different amounts of error to the total system. It can be determined from the individual error plots which sensors are contributing the most error and what improvements to the system would create the most improvement in the solution. Using the Pinson15 with our MagNav test a basic demonstration of error budgeting using covariance analysis can be conducted. The results from these error budgets can be seen below.

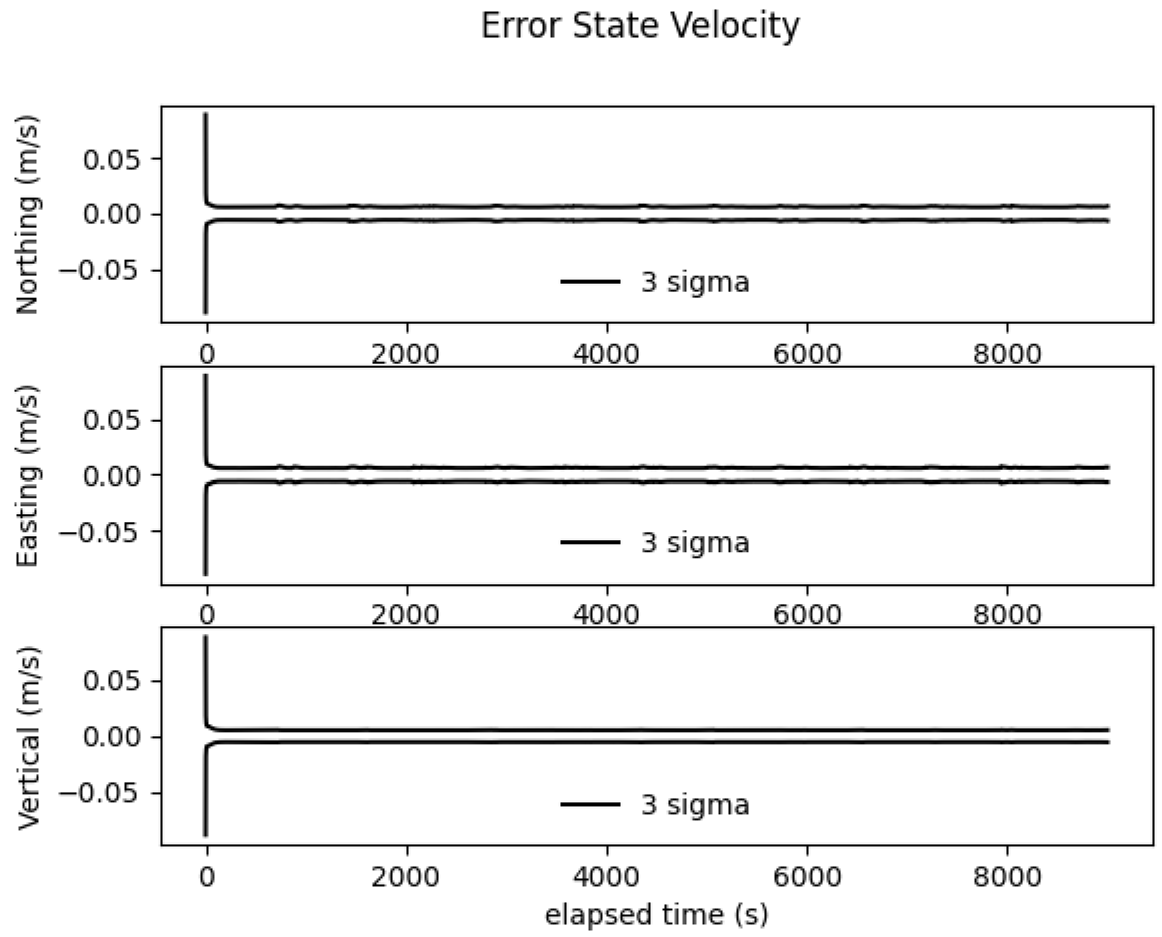


Figure 23: The velocity error state covariances are shown for the full simulation that includes the INS trajectory, velocity measurement update and MagNav measurement updates. As we expect, the velocity measurement stays within 3 centimeters per second, since we are measuring it directly.

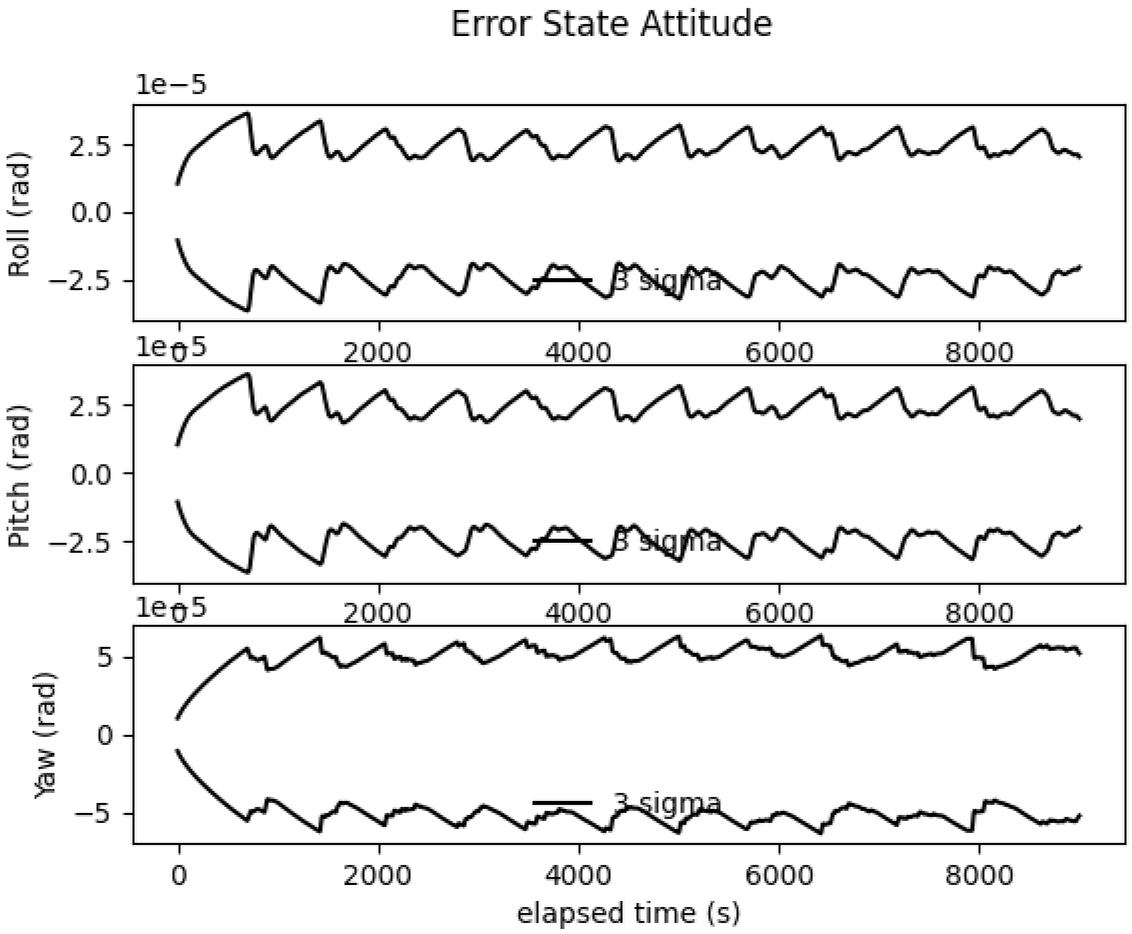


Figure 24: The attitude error state covariances are shown for the full simulation that includes the INS trajectory, velocity measurement update and MagNav measurement updates. Utilizing an HG9900 IMU.

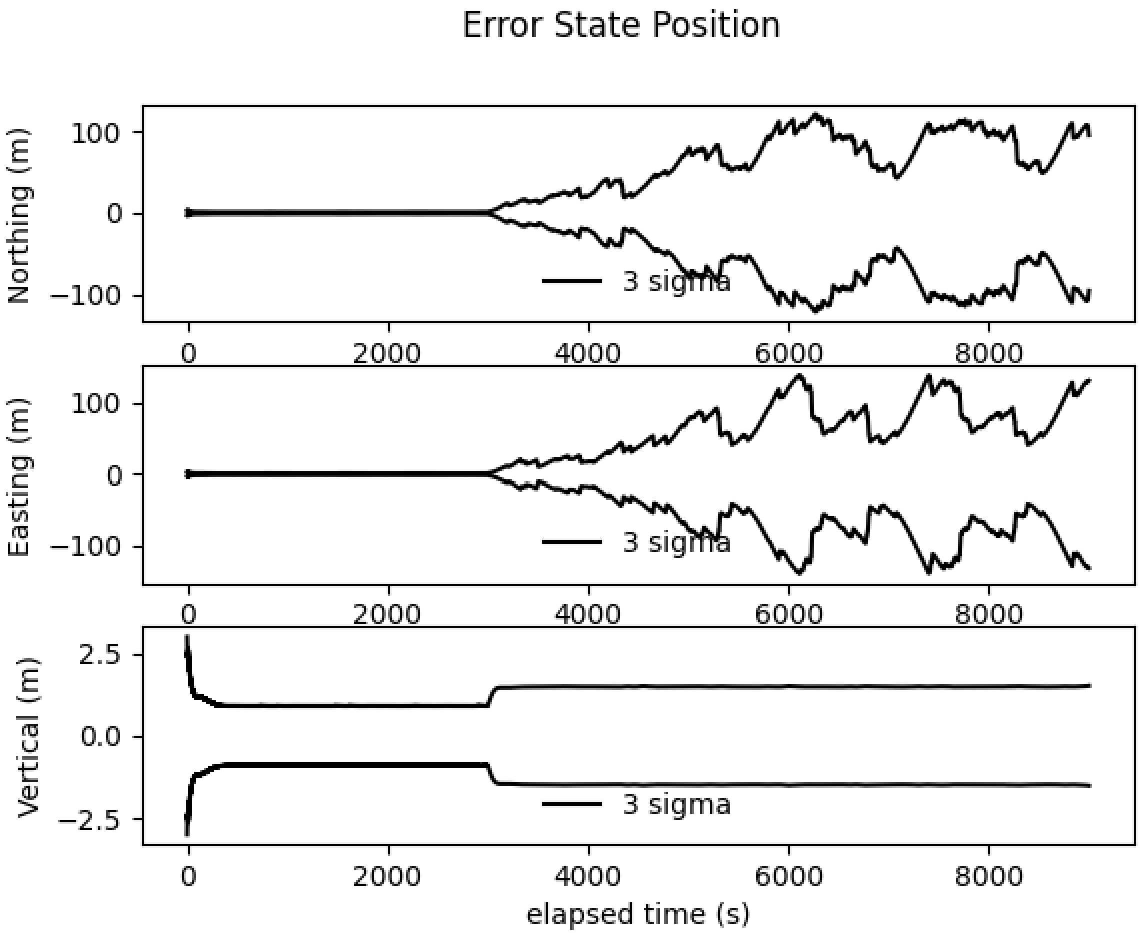


Figure 25: The position error state covariances are shown for the full simulation that includes the INS trajectory, MagNav measurement updates but excluding the Doppler LiDAR velocity sensor. 3-σ position error state covariance stays around 100$\pm$ meters

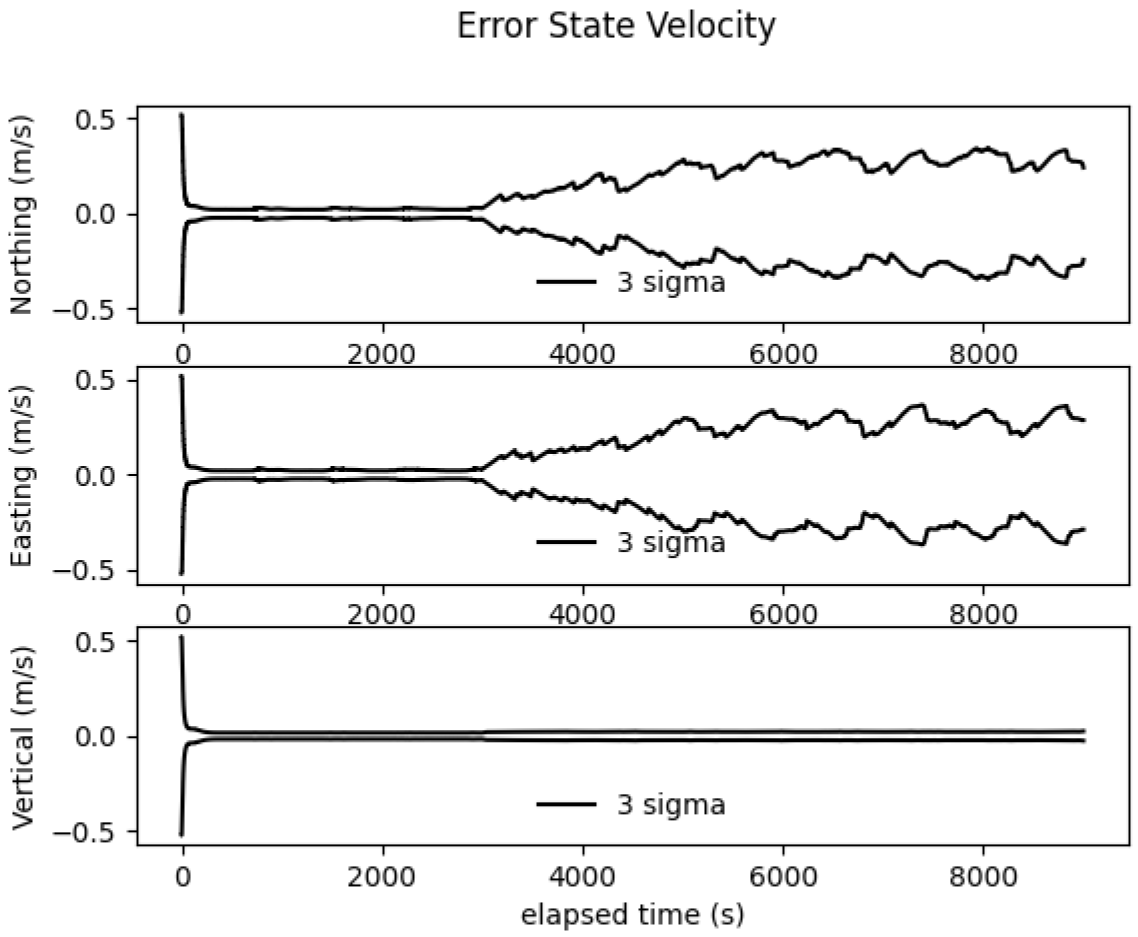


Figure 26: The velocity error state covariances are shown for the full simulation that includes the INS trajectory, MagNav measurement updates but excluding the Doppler LiDAR velocity sensor. 3-σ velocity error state covariance stays small, around .5$\pm$ meters per second.

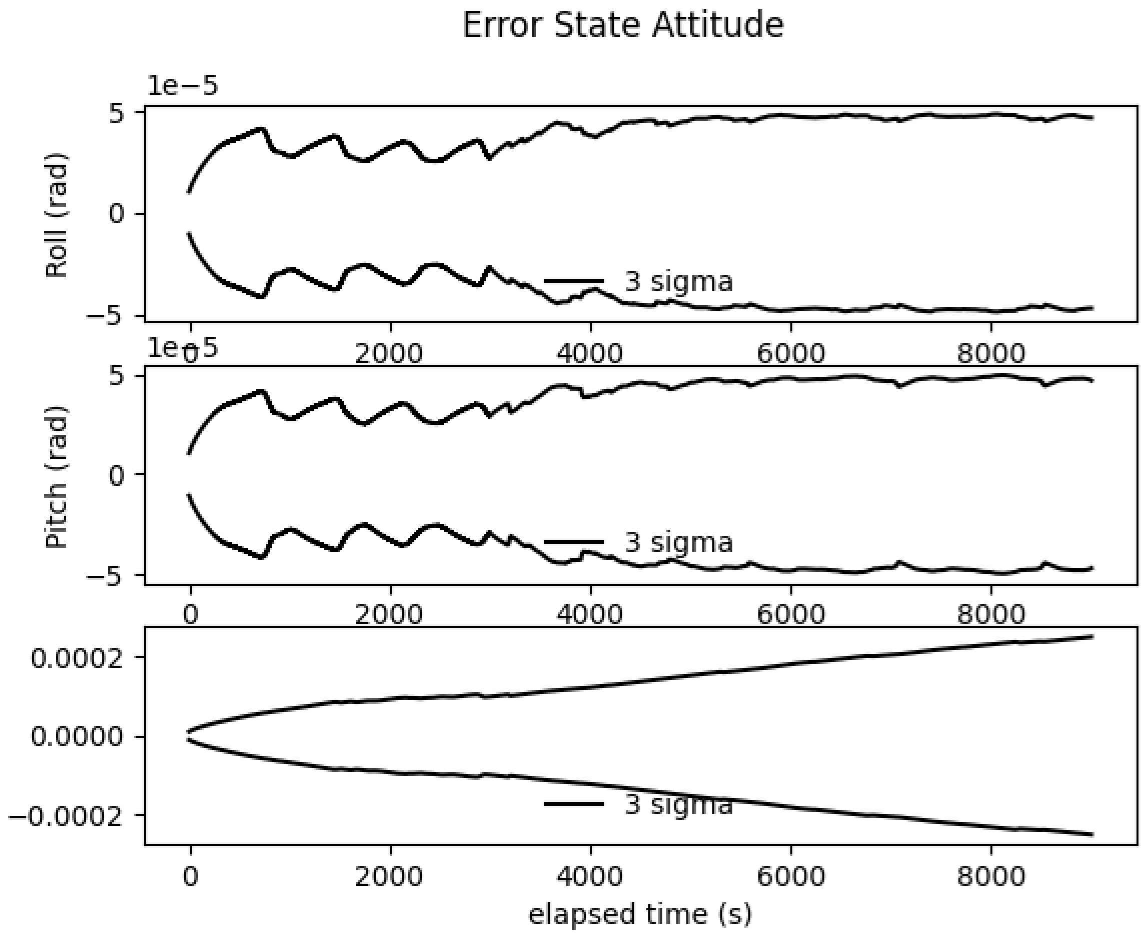


Figure 27: The attitude error state covariances are shown for the full simulation that includes the INS trajectory, MagNav measurement updates but excluding the Doppler LiDAR velocity sensor. Utilizing an HG9900 IMU.

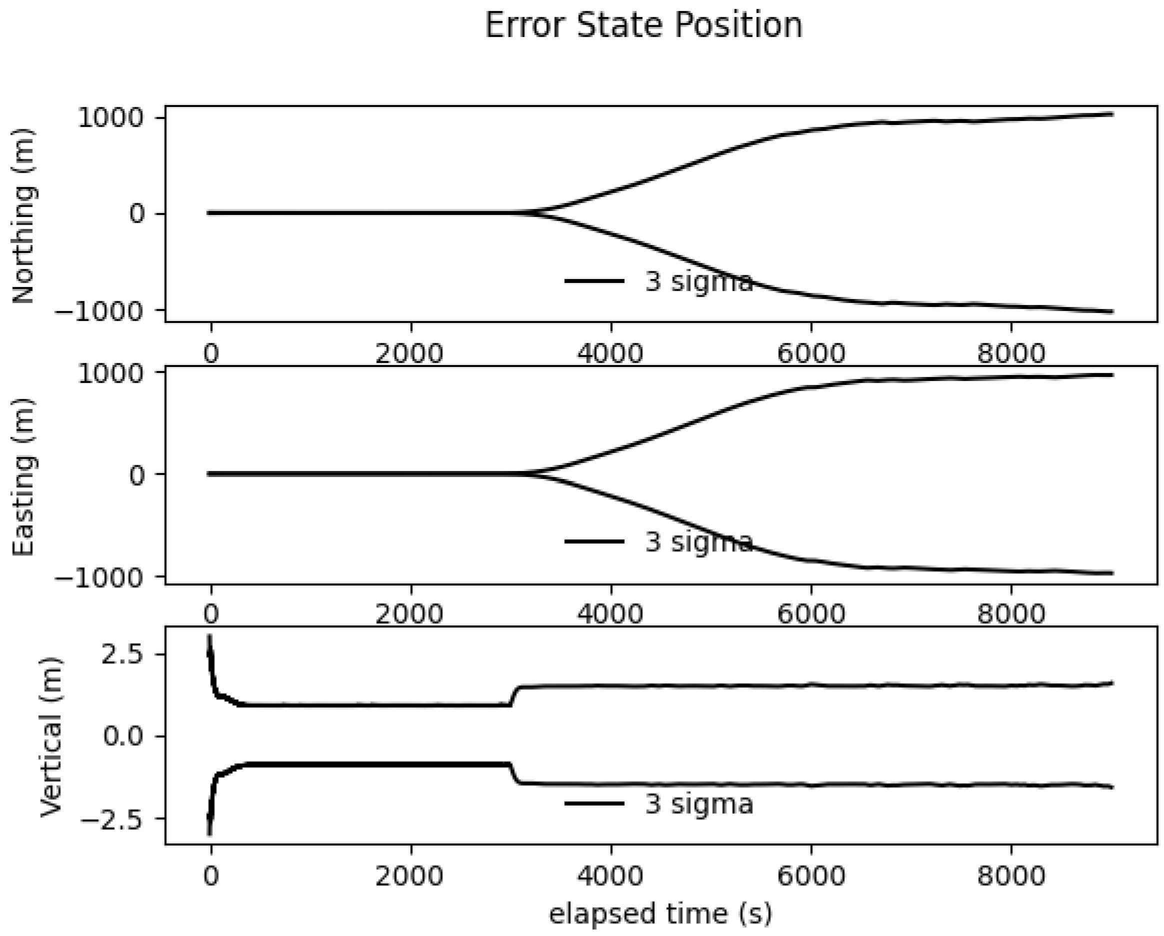


Figure 28: The position error state covariances are shown for the full simulation that includes the INS trajectory, excluding both the MagNav measurement updates and excluding the Doppler LiDAR velocity sensor. 3-σ position error state covariance stays around 1000± meters. This shows that with MagNav only, we cut the position error state covariance by ten times.

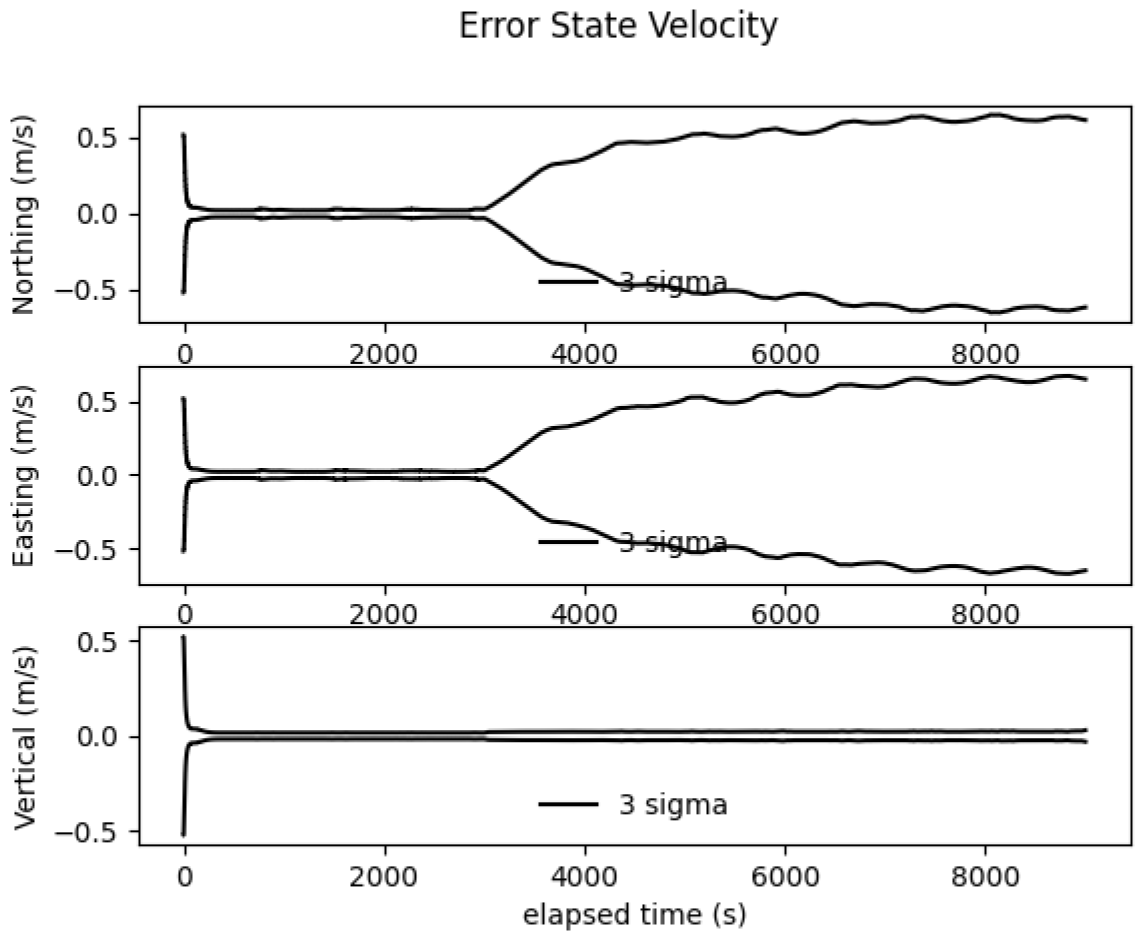


Figure 29: The velocity error state covariances are shown for the full simulation that includes the INS trajectory, excluding both the MagNav measurement updates and excluding the Doppler LiDAR velocity sensor. 3-σ velocity error state covariance stays small, around .5$\pm$ meters per second.

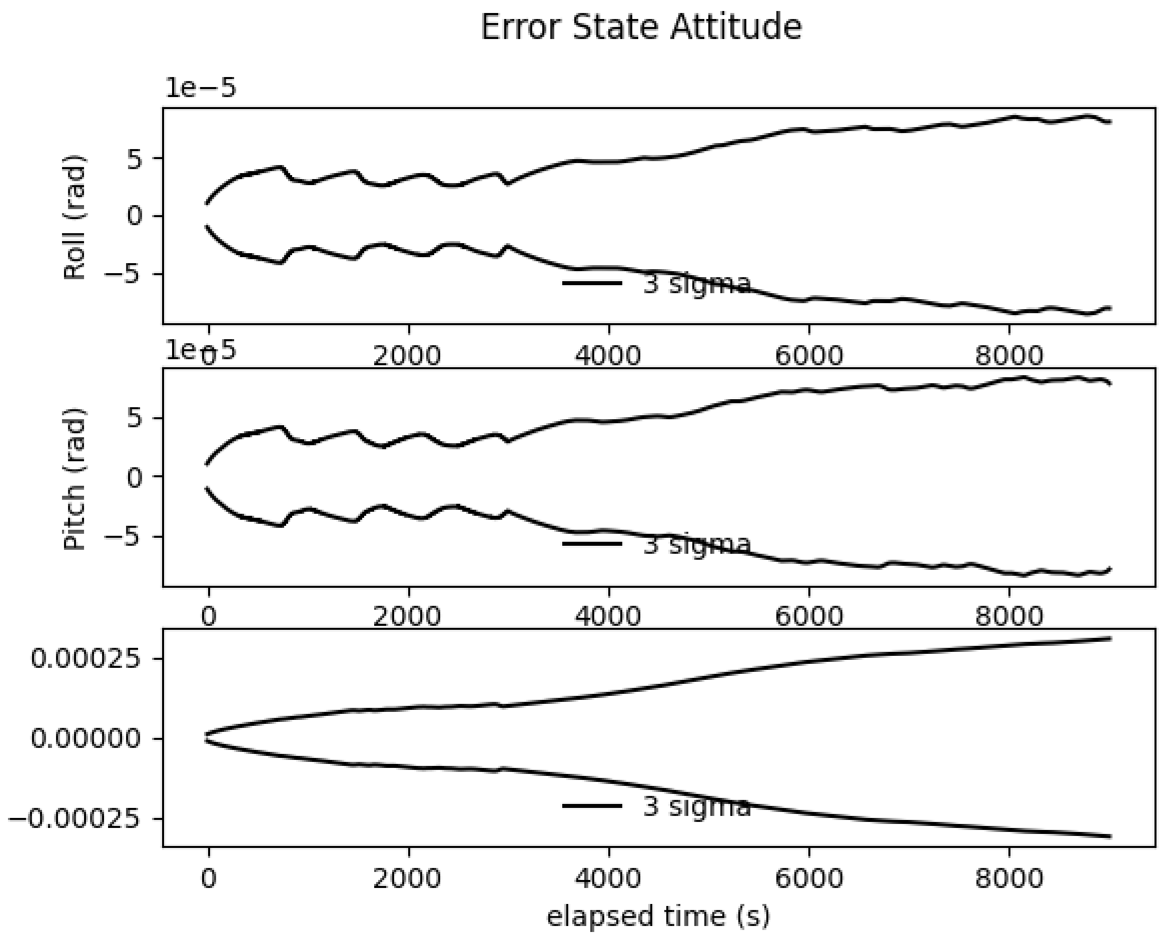


Figure 30: The attitude error state covariances are shown for the full simulation that includes the INS trajectory, excluding both the MagNav measurement updates and excluding the Doppler LiDAR velocity sensor. Utilizing an HG9900 IMU.

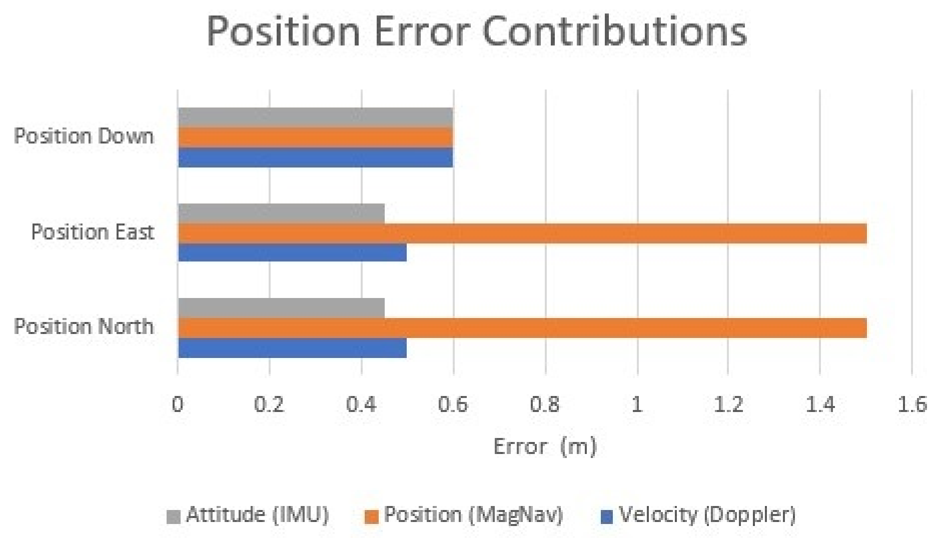


Figure 31: Error budgeting with positioning states. It can be concluded that a better positioning solution from our MagNav sensor will impact our overall positioning solution more than any other improvement.

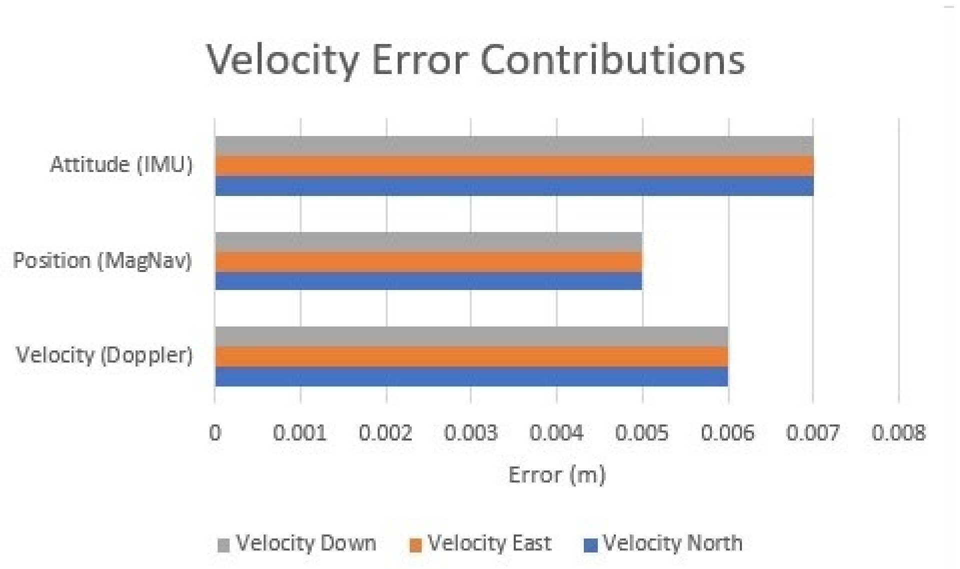


Figure 32: Error budgeting with velocity states. It can be concluded that a better velocity solution from our IMU sensor will impact our overall velocity solution more than any other improvement. The differences in these errors are very small and our velocity measurement accuracy is already very impressive so these improvements would be small in scale.

4.5 Conclusions

We have described a simulation environment that allowed the analysis of a GPS-alternative sensors and navigation solutions. We gave a brief intro to covariance analysis and Monte Carlo analysis. Then a description of the software developed and tool created to conduct covariance analysis. We finished with the data we obtained from the covariance analysis and an analysis of the results. We hope now with this tool and analysis performed we can give researchers an easy tool to use for trade-space analysis for navigation solutions. We also hope that we have helped outline the benefits and practicality of a magnetometer sensor. How a magnetometer sensor can aid in the positioning accuracy of an aircraft and also how it can act as a passive GPS-alternative when GPS may be unavailable. Trade-space analysis like this magnetic anomaly analysis is vital to exploration into new sensors and forms of navigation. With a thoroughly detailed and easy to use software package, researchers will be able to conduct this type of research with ease and be able to take advantage of the low cost covariance analysis simulation brings in the realm of time and monetarily. With more sensors added to the software framework over time and an ever-growing amount of developers/users, the versatility of the software framework will only grow. The effects of GPS drop outs in navigation solutions that included neither MagNav or Doppler LiDAR, only MagNav, then finally both MagNav and Doppler LiDAR were explored. We found that without MagNav nor Doppler LiDAR the loss of the GPS positioning solution led to a 1000$\pm$ meter error state covariance. Then, with MagNav added the simulation had a 100$\pm$ meter positioning state error. Finally, with MagNav and the Doppler LiDAR velocity sensor kept a steady 2.5$\pm$ meter error state covariance. This shows the great positioning solution that the fusion of Doppler LiDAR and MagNav can possibly bring in a navigation solution.

V. Conclusions

This final chapter discusses the future work on the covariance analysis tool. This chapter will also bring together the conclusions of the two individual papers in Chapter III and Chapter IV.

5.1 Use Case Conclusions

From the results discussed at the end of Chapter III and Chapter IV we have shown the possibilities of an accurate positioning solution when utilizing the Doppler Light Detection And Ranging (LiDAR) velocity sensor. In Chapter III we documented the results of a small test with the Doppler LiDAR velocity sensor that displayed sub-meter positioning solution over the 400 second simulation. Also, when coupled with Magnetic Anomaly Navigation (MagNav), we have shown how the Doppler LiDAR and MagNav can keep a very accurate positioning solution without GPS. The effects of GPS drop outs in navigation solutions that included neither MagNav or Doppler LiDAR, only MagNav, then finally both MagNav and Doppler LiDAR were explored. We found that without MagNav nor Doppler LiDAR the loss of the GPS positioning solution led to a 1000± meter error state covariance. Then, with MagNav added the simulation had a 100± meter positioning state error. Finally, with MagNav and the Doppler LiDAR velocity sensor kept a steady 2.5± meter error state covariance. This shows the great positioning solution that the fusion of Doppler LiDAR and MagNav can possibly bring in a navigation solution.

5.2 Covariance Analysis Tool Conclusions

The covariance analysis tool has shown through these use cases that it can produce results quickly and works well inside of the Position, Navigation, and Timing

Operating System (pntOS) framework. By documenting the GUI version of the tool alongside the code version readers and users can now conduct covariance analysis easily on navigation scenarios. This tool will allow researchers to quickly determine the effectiveness of difference sensors and determine which sensors affect a navigation solution more than others. The ease of exchanging sensors, by utilizing the pntOs framework, will allow researchers to determine which contribute the largest difference in covariance.

5.3 Future Work

This covariance tool utilizes an open-loop covariance analysis that is the most basic version of a covariance analysis. The covariance tool inside of the PntOS framework is limited in the amount of filters, algorithms, and models found inside the navigation toolkit. The expansion of the amount of filters, algorithms, and models is something that is always being worked on. Also, as we cited earlier, covariance analysis can be expanded to a closed-loop covariance analysis [7]. This allows analysis of hosted payloads and a much wider array of different navigation simulations.

Appendix A. Code Snippet

Shows where a user would input their own data when using the headless version of the covariance analysis tool.

```cpp
constexpr auto VELO_ENABLED = true;
constexpr auto POS_ENABLED  = false;
constexpr auto BARO_ENABLED = true;

//saves the output plot data into a CSV
constexpr auto OUTPUT_CSV_COVARIANCE = true;
//  */libviper/build is home directory this is will the output will save by default
constexpr auto OUTPUT_CSV_PATH = "output.csv";
/**path to truth to be used in simulation, default structure is 09,k7time, lat, long, alt, vN, vE, vD, R, P, Y
 you can change this structure inside of "parseCSV" however you want, you will just have to also modify "data_to_Navsolution"
 to account for your changes so the navsolution builds correctly
*/
constexpr auto PATH_TO_TRUTH = "/home/antuser/libviper/examples/TruthwRPY.csv";


// Turn feedback on/off
```

```
constexpr auto FEEDBACK_ENABLED = false;

// Some convenient constants (in seconds)
constexpr auto MINUTE = 60.0;

// If the source is enabled, choose the measurement
    sigma in meters
constexpr auto VELO_SIGMA = .003;
constexpr auto POS_SIGMA  = 1.0;
constexpr auto BARO_SIGMA = 1.0;

// If the source is enabled, this is the interval (
   sec) at which we receive measurements
// (NOTE: needs to be a multiple of DT)
constexpr auto VELO_INTERVAL = 1;
constexpr auto POS_INTERVAL  = 1 * MINUTE;
constexpr auto BARO_INTERVAL = 1;

// Choose interval to apply feedback
constexpr auto FEEDBACK_INTERVAL = 1 * MINUTE;

// How often we propagate our solution
constexpr auto DT = 1;

// Number of states in pinson block
constexpr auto NUM_STATES = 15;
```

```

        // Choose INS model for the Pinson state block

        // Navigation Grade
        auto model = viper::filtering::hg9900_model();

        //tactical grade
        //auto model = viper::filtering::hg1700_model();

        // Custom Model
        // auto model = viper::filtering::ImuModel{zeros(3) + 3e-3,   // accel_random_walk_sigma
        //   zeros(3) + 3e-5,   // gyro_random_walk_sigma
        //   zeros(3) + 1e-2,   // accel_bias_sigma
        //   zeros(3) + 3600,   // accel_bias_tau
        //   zeros(3) + 5e-6,   // gyro_bias_sigma
        //   zeros(3) + 3600};  // gyro_bias_tau
```

```

}
```

Bibliography

1. F. Landis Markley and J. Russell Carpenter. Generalized linear covariance analysis. *Journal of the Astronautical Sciences*, 57(1-2):233–260, 2009.

2. Paul J. Huxel and Babak E. Cohanim. Small lunar lander/hopper navigation analysis using linear covariance. *IEEE Aerospace Conference Proceedings*, pages 1–6, 2010.

3. Kyle Kauffman. pntos: Modular open software approach for real-time sensor fusion. *JNC 2021*, 2021.

4. Kyle Kauffman, Daniel Marietta, John Raquet, Daniel Carson, Robert C. Leishman, Aaron Canciani, Adam Schofield, and Michael Caporellie. Scorpion: A Modular Sensor Fusion Approach for Complementary Navigation Sensors. *2020 IEEE/ION Position, Location and Navigation Symposium, PLANS 2020*, pages 156–167, 2020.

5. Aaron Joseph Canciani and Christopher J. Brennan. An Analysis of the Benefits and Difficulties of Aerial Magnetic Vector Navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 56(6):4161–4176, 2020.

6. Peter Maybeck. *Stochastic Models, Estimation and Control: Volume 1*. Academic Press, 111 Fifth Ave, New York, 10003, 1979.

7. Randall S. Christensen and David K. Geller. Closed-loop linear covariance analysis for hosted payloads. *Journal of Guidance, Control, and Dynamics*, 41(10):2133–2143, 2018.

8. Aaron J. Canciani. Magnetic Navigation on an F-16 Aircraft using Online Calibration. *IEEE Transactions on Aerospace and Electronic Systems*, pages 1–15, 2021.

9. Jonnathan Bonifaz. Magnetic navigation using online calibration filter analysis, 2022.

10. Alex McNeil. Magnetic anomaly absolute positioning for hypersonic aircraft, 2022.

11. Robert C. Leishman and Tristan Williams. Validation of doppler lidar sensor using covariance analysis. *NAECON'21*, 2021.

12. Peter S Maybeck. The kalman filter: An introduction to concepts. In *Autonomous robot vehicles*, pages 194–204. Springer, 1990.

13. Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

14. Peter Maybeck. *Stochastic Models, Estimation and Control: Volume 2*. Academic Press, 111 Fifth Ave, New York, 10003, 1979.

15. David Titterton, John L Weston, and John Weston. *Strapdown inertial navigation technology*, volume 17. IET, 2004.

16. Randall S. Christensen and David Geller. Linear covariance techniques for closed-loop guidance navigation and control system design and analysis. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 228(1):44–65, 2014.

17. Randall Christensen and Robert C Leishman. Air Force Institute of Technology Real-Time Path Planning in Constrained , Uncertain Environments A key enabler

of autonomous vehicles is the ability to plan the path of the vehicle to accomplish mission Air Force Institute of Technology. 2019.

18. Eric W. Schoon, David Melamed, Ronald L. Breiger, Eunsung Yoon, and Christopher Kleps. Precluding rare outcomes by predicting their absence. *PLoS ONE*, 14(10):1–14, 2019.

19. David K. Geller. Linear covariance techniques for orbital rendezvous analysis and autonomous onboard mission planning. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference*, 1(August):424–444, 2005.

20. Adam Sievers, Renato Zanetti, and David C. Woffinden. Multiple event triggers in linear covariance analysis for spacecraft rendezvous. *AIAA/AAS Astrodynamics Specialist Conference 2010*, pages 1–34, 2010.

21. Kyle Kauffman. Modular open systems approach (mosa) to pnt systems and plug and play sensors: Modular gps independent sensors (mogis) project operational system demonstration. *JNC 2019*, 2019.

Acronyms

AFIT Air Force Institute of Technology. 15

ANT Center Autonomy and Navigation Technology Center. 15

CSV Comma-Separated Values. 23, 32, 37, 62

EKF extended Kalman Filter. 24, 32, 55, 61

GNC Guidance, Navigation, and Control. 29, 50

GPS Global Positioning System. 4, 28, 54, 65, 67

IMU inertial measurement unit. 3, 12, 23, 24, 27, 28, 37, 46, 51, 53, 62, 65

INS Inertial Navigation System. 20, 32, 57, 58

JVM Java Virtual Machine. 15

LiDAR Light Detection And Ranging. 3, 4, 27, 28, 32, 37, 46, 47, 48, 53, 57, 61, 65, 67, 79

MagNav Magnetic Anomaly Navigation. viii, 4, 23, 48, 54, 55, 61, 65, 67, 79

NAECON National Aerospace and Electronics Conference. 4

PNT Position Navigation and Timing. 16

pntOS Position, Navigation, and Timing Operating System. vii, 2, 15, 16, 18, 23, 55, 57, 61, 79

UI User Interface. 23, 62

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 24–03–2022	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Covariance Analysis for Multi-Source Navigation Architecture

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Tristan T. Williams

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-22-M-073

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Currently, analysis on navigation systems can be slow and computationally expensive using Monte Carlo approaches. Covariance analysis is a tool that can return trade space analysis results promptly and can be computationally reasonable. This research aims to create a covariance analysis tool in a new navigation framework architecture, PntOS. The creation of this covariance tool is explained in coordination with the tool being used in a few different navigation scenarios with the results. These scenarios include a Doppler LiDAR velocity sensor and magnetic anomaly navigation.

15. SUBJECT TERMS

covariance analysis, Monte Carlo analysis, pntOS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Tristan T. Williams, AFIT/ENG
U	U	U	UU	101	**19b. TELEPHONE NUMBER** *(include area code)* (937) 694-9110; tristan.williams.ctr@afit.edu