Spring 5-2022

Implementation of an Extended Kalman Filter Using Inertial Sensor Data for UAVs During GPS Denied Applications

Sky Seliquini
Old Dominion University, skyseliquini@gmail.com

IMPLEMENTATION OF AN EXTENDED KALMAN FILTER USING INERTIAL SENSOR DATA FOR UAVS DURING GPS DENIED APPLICATIONS

by

Sky Seliquini
B.S. December 2015, Florida Institute of Technology

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

ENGINEERING - AEROSPACE

OLD DOMINION UNIVERSITY
May 2022

ABSTRACT

IMPLEMENTATION OF AN EXTENDED KALMAN FILTER USING INERTIAL SENSOR DATA FOR UAVS DURING GPS DENIED APPLICATIONS

Sky Seliquini
Old Dominion University, 2022
Director: Dr. Thomas Alberts

Unmanned Aerial Vehicles (UAVs) are widely used across the industry and have a strong military application for defense. As UAVs become more accessible so does the increase of their applications, now being more limited by one's imagination as opposed to the past where micro electric components were the limiting factor. Almost all of the applications require GPS or radio guidance. For more covert and longer range missions relying solely on GPS and radio is insufficient as the Unmanned Aerial System is vulnerable to malicious encounters like GPS Jamming and GPS Spoofing. For long range mission GPS denied environments are common where loss of signal is experienced. For autonomous flight GPS is a fundamental requirement. In this work an advanced inertial navigation system is proposed along with a programmable Pixhawk flight controller and Cube Black autopilot. A Raspberry Pi serves as a companion computer running autonomous flight missions and providing data acquisition. The advancement in inertial navigation comes from the implementation of a high end Analog Devices' IMU providing input to an Extended Kalman Filter (EKF) to reduce error associated with measurement noise. The EKF is a efficient recursive computation applying the least-squares method. UAS flight controller simulations and calibrations were conducted to ensure the expected flight capabilities were achieved. The developed software and hardware was implemented in a Quadcopter build to perform flight test. Flight test data were used to analyze the performance post flight. Later, simulated feedback of the inertial navigation based state estimates (from flight test data) is performed to ensure reliable position data during GPS denied flight. The EKF applied to perform strapdown navigation was a limited success at estimating the vehicles' inertial states but only when tuned for the specific flight trajectory. The predicted position was succesfully converted to GPS data and passed to the autopilot in a LINUX based simulations ensuring autonmous mission capability is maintainable in GPS denied enviornments. The results from this research can be applied with ease to any vehicle operating with a Pixhawk controller and a companion computer of the appropriate processing capability.

To my family.

ACKNOWLEDGMENTS

I would like to begin with a very big thank you to Dr. Thomas Alberts who supported me through my research journey. His willingness to provide me with all the hardware as well as his mentorship played a huge role in the success of this research. I would especially like to thank Dr. Ahmed Mekky for his guidance that led to a better understanding of the EKF method used for the inertial state estimation making up the backbone of this effort. Additionally, I would like to give special attention to Dr. Duval and Mr. Robert Stewart. Brian Duval walked me through SITL applications and helped me to realize the possibilities attainable through Mavlink messaging and Dronekit. Mr. Stewart got me through a huge hurdle in acquiring data from the external IMU and allowed me to scavenge his drone supply to rebuild my damaged Quadcopter. He was also involved in most of the test flights conducted which were critical to analysis that was done throughout this research. Lastly, I would like to thank my loved ones who were a constant source of support and listened to me countless times go on and on about my never-ending thesis.

NOMENCLATURE

$\boldsymbol{EKF}$	Extended Kalman Filter
$\boldsymbol{PID}$	Proportional Integral Derivative
$\boldsymbol{LQR}$	Linear Quadratic Regulator
$\boldsymbol{UAV}$	Unmanned Arial Vehicle
$\boldsymbol{UAS}$	Unmanned Arial System
$\boldsymbol{GPS}$	Global Positioning System
$\boldsymbol{SITL}$	Simulation in The Loop
$\boldsymbol{MEMS}$	Micro Electro Mechanical Systems
$\boldsymbol{DOF}$	Degrees of Freedom
$\boldsymbol{INS}$	Inertial Navigation System
$\boldsymbol{GNSS}$	Global Navigation Satellite System
$\boldsymbol{IMU}$	Inertial Measurement Unit
$\boldsymbol{K_K}$	Kalman Gain
$\boldsymbol{E}$	Error
$\boldsymbol{\Delta T}$	Discrete Time Step
$\boldsymbol{\mathcal{N}}$	Probabilistic Distribution
$\boldsymbol{\mu}$	Probabilistic Mean
$\boldsymbol{\sigma^2}$	Probabilistic Variance
$\boldsymbol{\Sigma}$	Covariance Matrix
$\boldsymbol{\Psi}$	Yaw
$\boldsymbol{\Theta}$	Pitch
$\boldsymbol{\Phi}$	Roll
$\boldsymbol{R}$	Euler Rotation
$\boldsymbol{s_t}$	Translational Position Estimate
$\boldsymbol{v_t}$	Translational Velocity Estimate
$\boldsymbol{a_t}$	Translational Acceleration Estimate
$\boldsymbol{\delta}$	Rotational Attitude Estimate
$\boldsymbol{\omega}$	Rotational Velocity Estimate
$\boldsymbol{API}$	Application Program Interface
$\boldsymbol{Q}$	Process Noise Covariance Matrix
$\boldsymbol{R}$	Measurement Noise Covariance Matrix

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Over the last few decades UAVs have become more readily available and are no longer just military based tools but, can be used across all platforms ranging from civil applications to novice hobbyist. Smaller UAVs have especially increased in popularity with brands such as 3D Robotics, DJI, and PARROT, to name a few, saturating the market with easy to use drones. We have seen new economic titans such as Amazon and Verizon begin to dabble in the drone industry attempting to implement and exploit drone capabilities to streamline package deliveries or cell reception connectivity respectively. Flight controllers are a critical component of UAV hardware integration as they provide the autonomous operation capability. Flight controllers rely on stabilization and trajectory support systems made up of a suite of different sensors. This ensures reliable stabilization and robustness to sudden changes in the environment that may be unpredictable. However, stabilization is not a trivial task and the effort is exacerbated for small scale implementations, such as those used in small UAVs as the Micro Electro Mechanical Systems sensor are more compact, cheaper, and a consequence, much noisier. With the increase in drone based applications and complexity, an increase in system requirements is necessary to ensure safe autonomous flight.

Autonomous flight is based on the principles of guidance navigation and control. For UAVs' guidance or trajectory generation a path is determined based on the vehicle state, waypoints, mission objectives, avoidance maneuvers, target tracking, etc. [7]. Navigation is a skill going back to ancient times describing travel and finding the way from one place to another [8]. In modern day, it is a way to keep track of the system's state especially as it relates to an inertial frame i.e. position, speed, and attitude. Common sensors used for navigation are accelerometers and gyroscopes. Integrating the output from an accelerometer yields speed, and from integrating speed one can determine distance traveled. The gyroscope is necessary as it provides the direction of the accelerations and therefore a combination of the two yields heading and distance. A basic example of one dimensional navigation is illustrated in Figure 1. Control methodologies are used in ensuring the UAV remains on track and stays safe based on information from guidance and navigation. Historically, linear control has been applied linearizing the vehicles dynamics around a desired operation point with Proportional-integral-derivative (PID) and linear quadratic regulator (LQR) control methods. For example a simple hover and altitude control.[9] More complicated flight operations and improved performance are achieved using nonlinear control for the generalized form of the UAV's dynamics throughout the entire flight envelope. Some effective nonlinear control methods that are common among Quadrotor control are backstepping [10] and sliding mode [9].

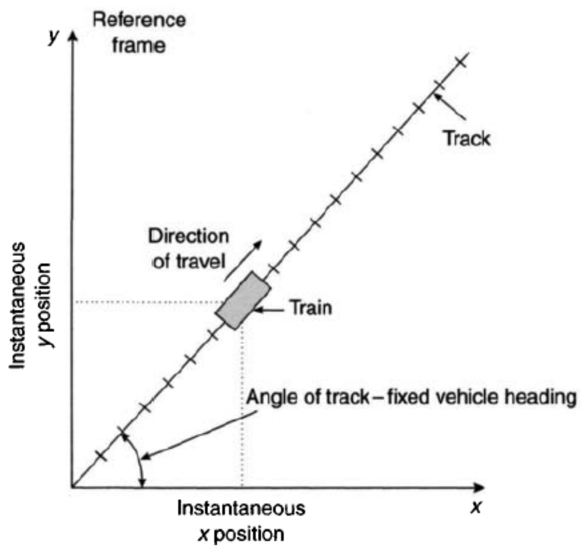


Fig. 1: One Dimensional Navigation of a Train

In some cases, a control system may have sufficient performance by relying solely on the state variables available for measurement. However, as to not limit design robustness, it is apparent this cannot be the only case, as if a system is observable, it is possible to estimate other states that may not be directly measured. An unforced system is said to be observable if and only if it is possible to determine any (arbitrary initial) state $x(t) = x_t$ by using only finite record, $y(\tau)$ for $t \leq \tau \leq T$, of the output[11]. Observers were first introduced by

Luenberger in 1963, applying them to linear systems for state estimation[12]. He showed that state estimation error could be minimized by various pole placements. R.E. Kalman with collaboration from R. Bucy developed an optimum state estimator with respect to process and observation noise[11]. The two key ingredients for the optimal state observer was that the system must be linear and the distributions must be Gaussian. The proposed method was known as the Kalman Filter. State estimators are invaluable when it comes to UAV auto pilots as they make it possible to obtain information of an otherwise unmeasured state. This work focuses primarily on the application of the Extended Kalman Filter state estimator but, works [13] and [14] may be referred to for more information.

There are several state of the art controllers and autopilots varying in design applications and quality. Flight controller hardware is readily available, designed to serve the novice drone hobbyist as well as the department of defense. It goes without saying that the capability and objectives are significantly different when it comes to military applications. For Quadcopter design, a popular flight controller/autopilot combination is the Pixhawk Cube autopilot. The Pixhawk series is an open source hardware flight controller that runs the autopilot firmware. APM(ArduPilot) firmware is capable of maintaining the baseline controllability of a UAV, in this case a Quadcopter. It offers various flight modes corresponding to different levels of UAV stability, for example it can enable altitude hold mode (ALT_HOLD) or stabilize mode (STABILIZE). In STABILIZE mode the Quadcopter remains level and ALT_HOLD mode the throttle is automatically maintained. The Cube Black is a Pixhawk autopilot that further evolves the Pixhawk flight controller offering an additional suite of on-board sensors and offers a more robust design running Ardupilot. The Cube Black is designed to work with commercial systems and other manufacturers who wish to integrate a more complex autopilot into their system. It offers redundant IMUs and more advanced CPUs. The Cube is a great option for UAV flight control research and development. More details of the Cube autopilot and its applications for this research can be found in Section 4 of this document.

Unfortunately, with the rapid advancement in UAV subsystems and accessibility, there has also been an increased requirement for UAV security. Unmanned aerial systems (UASs) can to easily be hacked and compromised. While this may not be a problem for UAVs used in manual mode, it is critical for autonomous operations. Any transmission being sent to or received by a UAS, if compromised, can pose as a substantial threat to the UAV's safety and successful operation. Not only must GPS jamming and spoofing be considered but, the more commonly occurring, GPS blackout. The ability to still maintain autonomous flight in GPS denied environments is constantly being studied. Today there are many new sensors that are being implemented in the loop to better estimate localization and assist in position control. To mention a few, some new technologies being implemented are LIDAR sensors, optical flow sensors, Sonar sensors, and higher quality cameras attached to motorized gimbals. Nonetheless, being able to successfully maintain flight in a GPS-denied environment while using the UAS to accurately predict the orientation and position of the UAV is still one of the greatest challenge faced today.

This research aims to develop, implement, and test a state estimator for UAVs in GPS denied environments so that autonomous flight may be maintained. A Pixhawk Cube Black autopilot will be used to develop a Quadcopter with external high quality IMU sensors. The redundant IMUs included in the stock flight controller are cheaper and of much lower quality. A key issue with relying on an IMU for inertial position estimation is the inherent noise and biases experienced by the sensors. The drift compounds overtime making it not ideal for continuous flight. The proposed solution being researched is the implementation of an Extended Kalman Filter to reduce the error associated within the IMU based dead reckoning navigation. Software in the loop (SITL) ArduPilot is utilized to develop autonomous flight missions and data taking routines relying on Mavlink based messaging.

After achieving the desired performance in the SITL simulations a Quadcopter was built, designed with the processing capabilities required for the EKF implementation, external high end IMU hardware integration, and data acquisition. Arduino's IDE was utilized to develop a data routine to export data from the internal sensors of the Analog Devices IMU in a serial fashion through hardware in the loop simulations. After fully integrating the necessary hardware and software, autonomous flight tests were performed. Conclusions are drawn analyzing the EKF's performance based on the flight test data. As a proof of concept inertial based position estimates are relayed as Mavlink messages to the Quadcopter in an SITL simulation. The messages act as substitute for an on board GPS sensor, effectively taking it out of the loop. Finally, future works are discussed to improve upon the design implementation as well as the potential for a machine learning algorithm based intrusion detection system to monitor and classify GPS.

CHAPTER 2

LITERATURE REVIEW

2.1 QUADCOPTERS

A Quadcopter is a rotorcraft with 4 rotors and propellers capable of vertical take off. The first attempts at vertical take off were performed in the early 1900s with multirotor, a term used to describe any configuration with more than one rotor (tricopters, hexcopters, octocopters etc.), rotorcraft. In 1920, Etienne Oehmichen designed a rotorcraft with a steel tube frame design, four rotors, and 8 propellers (four arms with two bladed rotors at their ends). After thousands of test flights in 1924 the French engineer completed the first 1km range flight. Original Quadcopter designs had the engine placed centrally in the fuselage providing power to the rotors with belts or shafts [15]. Some main disadvantages of the design, however, were that belts and shafts are heavy and prone to breaking. More importantly, a Quadcopter is naturally unstable as simply spinning all the propellers at the same rate does not produce stable flight. As a result, eventually Quadcopter designs were scrapped and designers went with the more inefficient single main rotor design, known today as helicopters. A helicopter's tail rotor consumes 10 to 15 percent of the engines power while providing zero lift or forward thrust. It was not until the arrival of modern electric motors and Micro-Electro-Mechanical System (MEMS) that practical, efficient, and reliable Quadcopter builds became possible.

A Quadcopter has four rotary wings along with four electric motors, usually placed directly underneath the rotors. Typically each propeller spins at a different speed depending on the flight mode (hover, forward motion, vertical take off, etc.). The quad rotor setup is a 6 degrees of freedom (DOF) system with four control inputs and six outputs. The six outputs are roll, pitch, and yaw making up the vehicles attitude while the position is defined as the direction along the x, y, and z axes [16]. The only control inputs are the individual angular velocities of the four rotors. Given that the number of inputs is less than then number of outputs, the Quadcopter can be categorized as an under actuated nonlinear system. This makes it a great candidate for research into highly nonlinear under actuated systems and automatic control [9].

2.2 INERTIAL NAVIGATION SYSTEMS

An Inertial Navigation System (INS) is a critical component of any UAV's autopilot. An INS uses an onboard computer with a precision clock for integration and time step operations, an assembly of accelerometers measuring multidirectional acceleration force, and a suite of softwares that model the gravitational acceleration as a function of the calculated position. It also includes a reference of the vehicles attitude that is essential in describing the angular orientation of the orthogonally configured accelerometers that make up a part of the velocity calculations. In a modern-day INS, the physical attitude reference is replaced with a reference embedded in the modeling software in the onboard computer. It completes the integration from a set of three-axis inertial sensors measuring angular-rate. The gyroscope sensors along with the triad of accelerometers are mounted on a shared rigid structure within the chassis of the INS placed to ensure a precise alignment between the two is well maintained. This configuration denoted by the rigid attachment to the vehicle is known as a strapdown INS [17].

The primary functions executed in the INS computer are the angular rate into attitude integration function (denoted as attitude integration), use of the attitude data to transform measured acceleration into a suitable navigation coordinate frame where it is integrated into velocity (denoted as velocity integration), and integration of the navigation frame velocity into position (denoted as position integration). Thus, three integration functions are involved, attitude, velocity, and position, each of which requires high accuracy to assure negligible error compared to inertial sensor accuracy requirements. Savage discussed a rigorous comprehensive approach to the design of the principal software algorithms utilized in modern-day strapdown inertial navigation systems [18] [17].

As unmanned aircraft systems (UAS) operations across the globe continue to grow at an accelerated rate so do the posed new technological and regulatory challenges arise. INS are evolving into more robust navigation and guidance systems (NGS) encompassing more than just inertial measurement hardware. State of the art cameras and LIDAR are being integrated for localization, obstacle detection, and avoidance. [19] presents a comprehensive review of conventional UAS navigation systems, including aspects such as system architecture, sensing modalities, and data-fusion algorithms. The primary focus is on the identification of key gaps in the literature where the use of AI-based methods can potentially enhance navigation performance. A detailed review of Unmanned Aircraft System (UAS) intelligent navigation systems was presented, including identification and detailed explanations and analyses of conventional equipment and algorithms. It was noted that there are two types of integrity monitoring systems on which Global Navigation Satellite System's (GNSS) rely. The first of which is satellite redundancy that is less reliable in a more urban environment as it is intermittent. The second is a differential technique characterized as having the ability to detect and isolate ranging faults, however it is not optimal for multi-path trajectories. Typically, the inertial sensors used as inputs for integration in low SWAP-C navigation systems are low-cost Micro Electro-Mechanical System(MEMS)-IMUs. These low-cost sensors are known to be relatively noisy and rapidly diverge in their measurements within $20 - 30$ seconds. In most cases they alone cannot produce reliable position updates or compensate for their internal bias. Multi-sensor integrity monitoring algorithms have little application when dealing with these types of sensors as the noise levels lead to small measurement errors. The widespread presence of GNSS ranging errors in congested metropolitan like areas render missions in such areas unfeasible without major changes to the execution of the associated procedures. One example of a new technology that may alleviate such limitations is Vision-Based Navigation (VBN) which is now a mature field of research that supports accurate position predictions suitable for GNSS denied/degraded conditions. However the system integrity and robustness is not as well known given that decades of research and application development has gone into the traditional GNSS/INS integrated systems.

2.2.1 MEMS

It wasn't until the 1990s, with the development of Micro-Electro-Mechanical System (MEMS), that Inertial Measurement Units(IMUs) weighing several grams emerged. Although MEMS sensors have been designed, the low-cost MEMS IMUs produce large amounts of noise. Therefore the measurements they produce cannot be used directly. The research started to receive more and more attention on how to address the noise in the attitude measurement of MEMS IMUs. The design of a small multicopter requires not only algorithms but also microcomputers on which these algorithms can run. IMUs include: three-axis accelerometer, three-axis gyroscope, and an electronic compass (or three-axis magnetometer). It is used to obtain attitude information of a multicopter. In general, a six-axis IMU is the combination of a three-axis accelerometer and a three-axis gyroscope; a nine-axis IMU is the combination of a three-axis accelerometer, a three axis gyroscope and a three-axis magnetometer; and a ten-axis is the combination of a nine-axis IMU and a barometer [18].

2.2.2 IMU CALIBRATION

Today many low cost Micro Electro Mechanical Systems (MEMS) based IMU are available off the shelf, while smart phones and similar devices are almost always equipped with low-cost embedded IMU sensors. Nevertheless, low cost IMUs are affected by systematic error given by imprecise scaling factors and axes misalignment that decrease accuracy in the position and attitudes estimation. In [20], a robust and easy to implement method to calibrate an IMU without any external equipment is proposed. The procedure is based on a multi-position scheme, providing scale and misalignment factors for both the accelerometer and gyroscope triads, while estimating the sensor biases. The method only requires the sensor to be moved by hand and placed in a set of different, static positions (attitudes). The process can be described as a robust and quick calibration protocol that exploits an effective parameterless static Filter to reliably detect the static intervals in the sensor measurements, where local stability of the gravity's magnitude and temperature are assumed [20].

2.2.3 GPS

The Global Positioning System (GPS), shown as in Figure 2, is a GNSS that uses satellites to provide locations and time. The satellites use atomic clocks which are synchronized to each other, and the time of clocks is corrected by the true time of the ground clocks. GPS satellites' locations are monitored precisely. GPS receivers have clocks that are synchronized to satellite time. While GPS satellites broadcast their position and time, GPS receivers can get signals of multiple satellites. Thus, GPS receivers can calculate the exact position by solving equations, and the deviation can be eliminated as well. The accuracy of GPS is generally in meters. The GPS observations are affected by different factors: (1) satellite related, including orbital errors and satellite clock errors; (2) propagation errors, including the ionospheric delay error, tropospheric refraction error, and multipath errors; (3) the receiver error, including the receiver clock error and observation errors [18].

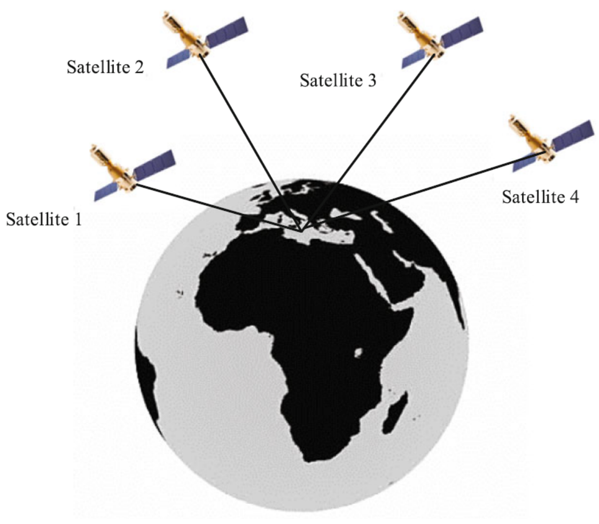


Fig. 2: Global Positioning System Representation

2.3 EXTENDED KALMAN FILTER

Before diving into the Extended Kalman Filter it it necessary to introduce its foundation, the Kalman Filter. In 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman Filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. The Kalman Filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. It supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown [1].

The Kalman Filter addresses the problem of trying to estimate the state $x \in \Re^n$ of a discrete time system that is governed by the linear stochastic difference equation

$$x_{k+1} = A_k x_k + B u_k + w_k, \tag{1}$$

in which a measurement $z \in \Re^m$ is

$$z_k = H_k x_k + v_k. \tag{2}$$

The variables w_k and v_k represent process noise and measurement noise respectively. They are assumed to be independent of each other and characterized as white noise with a normal probability distribution. Equation (1) relates the $n \times n$ matrix A at time step k to the state at time step $k+1$. Matrix B is size $n \times l$ and related the control input $u \in \Re$ to the state x. The $m \times n$ matrix H in Equation (2) relates the state to the measurement, z_k.

Priori and posteriori estimate errors are defined as the difference between the state (x_k) and the priori state estimate $(\hat{x}_k^-)$ and the difference between the state and the posteriori state estimate $(\hat{x}_k)$ at the time step of the given measurement respectively. Therefore the priori estimate error covariance is

$$P^-{}_k = E[e^-{}_k e^-{}_k^T], \tag{3}$$

and the posteriori estimate error covariance is

$$P_k = E[e_k e_k^T]. \tag{4}$$

In seeking the derived equations for the Kalman Filter, one can begin with the goal of finding an equation that computes the posteriori state estimate as a linear combination of the the priori estimate and the weighted difference between the measurement being observed and the measurement prediction. This relationship is described below in Equation (5)

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-). \tag{5}$$

The difference $K(z_k - H_k \hat{x}_k^-)$ in Equation (5) is referred to as the measurement innovation or residual and reflects the deviation between the measurement and the predicted state. The $n \times m$ matrix K in Equation (5) is the Kalman gain which is a balancing factor that minimizes the posteriori error covariance, Equation (3). The minimization can be accomplished by substituting Equation (5) into the definition of e_k and then substituting that into Equation (4) and performing the operation, taking the derivation with respect to K, setting the result equal to zero and solving for K. The final solution reduces to

$$K_k = \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + R_k}. \tag{6}$$

It is important to note that as K_K approaches zero the measurement is trusted less and less and ultimately the operation becomes solely based on the predicted state.

The Kalman Filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman Filter fall into two groups: time update equations and measurement update equations. The first task during the measurement update is to compute the Kalman gain, K_k. Next, measure the process to obtain z_k and then generate a posteriori state estimate by incorporating the measurement as in Equation (5). The final step is to obtain a posteriori error covariance estimate, P_k defined as

$$P_k = (I - K_k H_k) P_k^-. \tag{7}$$

After each time and measurement update pair, the process is repeated with the previous posteriori estimates used to project or predict the new priori estimates. This recursive nature is one of the very appealing features of the Kalman Filter—it makes practical implementations practicable[1]. When implementing the Filter, the process and measurement noise have error covariance matrices, $E(w)$ Q_k and $E(v)$ R_k. These parameters can often be tuned to impact performance. A complete overview of the Kalman Filter process is given below in Figure 3.

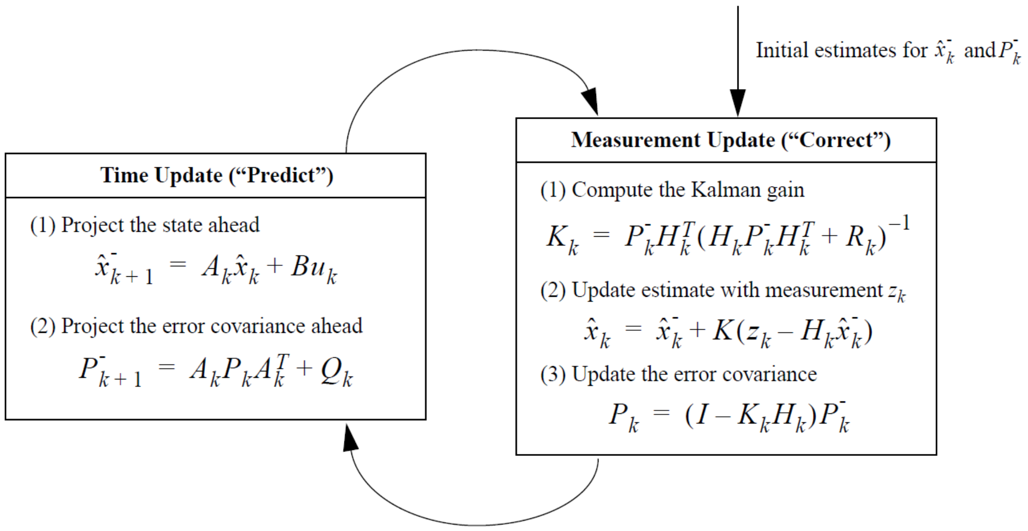

Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = A_k \hat{x}_k + B u_k$
(2) Project the error covariance ahead
$P_{k+1}^- = A_k P_k A_k^T + Q_k$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - H_k \hat{x}_k^-)$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$

Fig. 3: An Overview of Kalman Filter [1]

To conclude, the Kalman Filter tackles the general problem of trying to estimate the state, $x \in \Re^n$, of a discrete time controlled process via a linear stochastic difference equation (1). However, if the process to be estimated or the measurement relationship to the process is nonlinear this will not suffice. Que, the Extended Kalman Filter (EKF). An EKF linearizes about the current mean and covariance to estimate a process with nonlinear difference and measurement relationships. The process is now governed by a nonlinear stochastic difference equation

$$x_{k+1} = f(x_k, u_k, w_k), \tag{8}$$

where measurement $z \in \Re^m$ is

$$z_k = h(x_k, v_k). \tag{9}$$

The variables w_k and v_k represent process and noise again as in Equations (1) and (2). The nonlinear function $f(\bullet)$ within the nonlinear stochastic difference equation relates the state at time step k to the state at time step $k+1$. It includes the input function u_k and the zero mean process noise. The nonlinear function $h(\bullet)$ relates the state to the measurement. As the individual values for noise are unknown, the state and measurement can be approximated without them as described below. $\hat{x}_k$ is a posteriori estimate of the state from a previous time step.

$$\tilde{x}_{k+1} = f(\hat{x}_k, u_k, 0), \tag{10}$$

$$\tilde{z}_k = h(\tilde{x}_k, 0). \tag{11}$$

Now that the nonlinear stochastic difference relationship has been presented, the EKF can be further explained starting with new governing equations that linearize an estimate about equations (10) and (11).

$$x_{k+1} \approx \tilde{x}_{k+1} + A(x_k - \hat{x}_k) + W w_k, \tag{12}$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + V v_k, \tag{13}$$

where,

- x_{k+1} and z_k are the state and measurement vectors.

- $\tilde{x}_{k+1}$ and $\tilde{z}_k$ are the approximate state and measurement vectors seen in Equations (12) and (13).

- $\hat{x}_k$ is a posteriori estimate of the state at time step k.

- w_k and v_k are process and measurement noise as previously stated.

- A is a Jacobian matrix of partial derivatives of $f(\bullet)$ with respect to the state x,

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_k, u_k, 0). \tag{14}$$

- W is a Jacobian matrix of partial derivatives of $f(\bullet)$ with respect to the process noise w,

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_k, u_k, 0). \tag{15}$$

- H is a Jacobian matrix of partial derivatives of $h(\bullet)$ with respect to the measurement z,

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_k, 0). \tag{16}$$

- V is a Jacobian matrix of partial derivatives of $h(\bullet)$ with respect to the measurement noise v,

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\tilde{x}_k, 0). \tag{17}$$

Note, that the time step subscript, k, was not included in the Jacobian equations for simplicity as they are present in reality and change at each time step. A new notation for the prediction error and measurement residual are as follows:

$$\tilde{e}_{x_k} \equiv x_k - \tilde{x}_k \tag{18}$$

$$\tilde{e}_{z_k} \equiv z_k - \tilde{z}_k \tag{19}$$

Being that the physical state is not accessible, Equations (18) and (19) can be used to develop new governing equations for process error given that the measurement z_k is available. ε_k and η_k are new independent process and measurement noise variables having a mean of zero. The covariance matrices are respectively WQW^T and VRV^T

$$\tilde{e}_{x_{k+1}} \approx A(x_k - \hat{x}_k) + \varepsilon_k \tag{20}$$

$$\tilde{e}_{z_k} \approx H\tilde{e}_{x_k} + \eta_k \tag{21}$$

It becomes apparent that the linear Equations (20) and (21) share a resemblance to those originally introduced with the Kalman Filter, Equations (1) and (2). Therefore the measurement residual, $\tilde{e}_{z_k}$ along with a second Kalman Filter may be used to estimate the prediction error described in Equation (20). The new estimate, $\hat{e}_k$, along with Equation (18) are used to predict a new posteriori state estimate of the original nonlinear process resulting in,

$$\hat{x}_k = \tilde{x}_k + \hat{e}_k \tag{22}$$

$$p(\tilde{e}_{x_k}) \approx N(0, E[\tilde{e}_{x_k}\tilde{e}_{x_k}^T]) \tag{23}$$

$$p(\varepsilon_k \approx N(0, WQW^T) \tag{24}$$

$$p(\eta_k \approx N(0, VRV^T) \tag{25}$$

The random variables of the new governing equations for the error in the process all have the following probability distributions. It is worth mentioning that this is the fundamental flaw of the EKF. The probability distribution of the random variables are no longer normal after undergoing the nonlinear transformation. After applying the approximations above and setting the predicted value of $\hat{x}$ to 0, the Kalman Filter used to estimate $\tilde{e}_k$ is equal to $K_k\tilde{e}_{z_k}$. Substituting it back into Equation (22) and applying Equation (19) it is determined that a second Kalman Filter is not required as,

$$\hat{x}_k = \tilde{x}_k + K_k\tilde{e}_{z_k} = \tilde{x}_k + K_k(z_k - \tilde{z}_k). \tag{26}$$

Now Equation (26) may be used as the measurement update in the EKF where $\tilde{x}$ and $\tilde{z}_k$ are represented by the nonlinear stochastic difference equations in (12) and (13). The Kalman gain is K_k as defined in our linear approach, Equation (6), with substitutions for the newly linearized measurement error covariance. The complete list of EKF equations as derived above are presented in a process overview diagram given in Figure 4. A feature of the EKF that is of great consequence is that of the Jacobian matrix, H_k, used in the equation for the Kalman gain. It serves to correctly propagate or magnify only the relevant component of

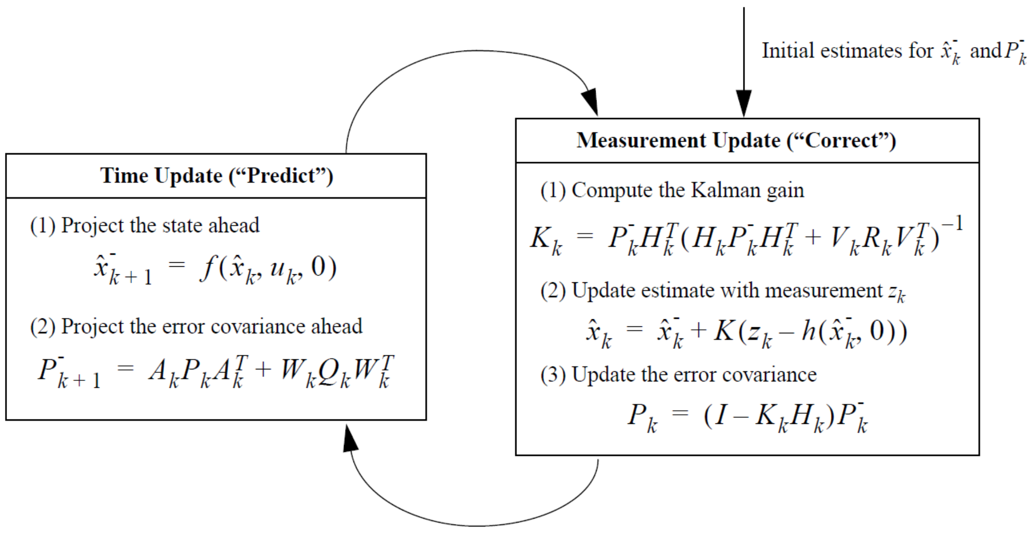

Initial estimates for $\hat{x}_k^-$ and P_k^-
Time Update ("Predict")
(1) Project the state ahead
$\hat{x}_{k+1}^- = f(\hat{x}_k, u_k, 0)$
(2) Project the error covariance ahead
$P_{k+1}^- = A_k P_k A_k^T + W_k Q_k W_k^T$
Measurement Update ("Correct")
(1) Compute the Kalman gain
$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1}$
(2) Update estimate with measurement z_k
$\hat{x}_k = \hat{x}_k^- + K(z_k - h(\hat{x}_k^-, 0))$
(3) Update the error covariance
$P_k = (I - K_k H_k) P_k^-$

Fig. 4: An Overview of Extended Kalman Filter [1]

the measurement information. If the system is not observable, i.e., there is not a one-to-one mapping of the measurement z_k and the state via $h(\bullet)$ then the EKF will quickly diverge and prove ineffective [1].

2.3.1 EKF FOR UAVS

[21] discusses the use of an EKF for UAV localization. Many smaller UAVs rely solely on GPS for navigation and unfortunately, it is not uncommon for GPS outages to occur even in benign environments. [21] proposes an EKF to estimate location during GPS connection loss using inter-UAV distance measurements. This application is geared towards operations with groups of UAVs cooperating together to complete missions. To simplify their approach a 2D problem was considered. The discrete time domain dynamic model of the UAV used is shown in the Equations below in which the state variables x and y represent the UAV coordinates in the horizontal plane, η and ω represent the heading and angular speed accordingly, and v represents the ground speed.

$$x_{k+1} = x_k + v_k \Delta t_k cos\eta_k$$

$$y_{k+1} = y_k + v_k \Delta t_k sin\eta_k$$

$$\eta_{k+1} = \eta_k + \omega_k \Delta t_k$$

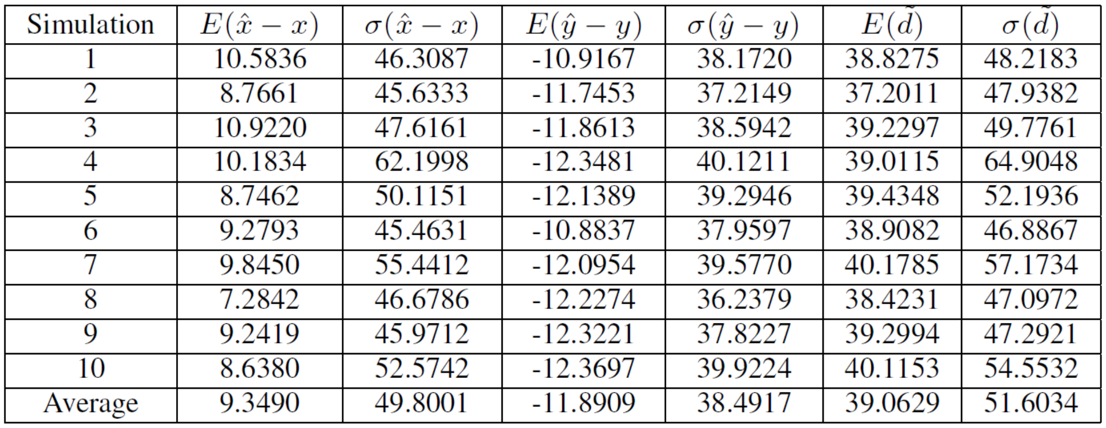
$$\omega_{k+1} = \omega_k + \epsilon_k$$

$$v_{k+1} = v_k + \epsilon_{v,k}$$

As the UAVs were not equipped to measure their inter-distances, real flight data that maintained GPS connectivity were used in a simulation in which a time series of synthetic GPS outages were applied. Ten simulations were completed where 9,718 location estimates were calculated.

Results were interpreted based off the mean error of the state estimates x and y, $E(\hat{x}-x)$ and $E(\hat{y}-y)$, the standard deviations (σ) of the error, and the mean value between the estimated location and the true GPS position. A summary of the results discussed in [21] are given in Figure 5. [21] verified that a range measurement to other UAVs can be used with an EKF in GPS-denied environments to perform location estimates accurate to within 40 meters of its true positions. The UAV separation applied in the experiment was approximately 5km, resulting in a range measurement error or 10 meters.

An extensive performance comparison of combinations of fusing accelerometer and gyroscope data as control or measurement inputs in an EKF is discussed in [22]. It is determined with simulated and real data that optimal performance is achieved when fusing both sensors in the measurement stage. The results definitively illustrate that accelerometer measurements are better for 3D position tracking accuracy and gyroscope measurements are ideal for 3D orientation accuracy. The major findings of the research was that both inertial sensors greatly improved 3D accuracy more when used as measurement inputs (MMM) when compared to all other combinations. Also, the improvement provided by gyroscope in position is more pronounced than the improvement provided by accelerometer in orientation, hence if only one inertial sensor is to be used, it should be gyroscope used as measurement. The result comparisons are displayed in Figure 6. More detail describing the mathematics of the optimal combination is presented in the Methodology section of this paper.



Simulation	$E(\hat{x}-x)$	$\sigma(\hat{x}-x)$	$E(\hat{y}-y)$	$\sigma(\hat{y}-y)$	$E(\tilde{d})$	$\sigma(\tilde{d})$
1	10.5836	46.3087	-10.9167	38.1720	38.8275	48.2183
2	8.7661	45.6333	-11.7453	37.2149	37.2011	47.9382
3	10.9220	47.6161	-11.8613	38.5942	39.2297	49.7761
4	10.1834	62.1998	-12.3481	40.1211	39.0115	64.9048
5	8.7462	50.1151	-12.1389	39.2946	39.4348	52.1936
6	9.2793	45.4631	-10.8837	37.9597	38.9082	46.8867
7	9.8450	55.4412	-12.0954	39.5770	40.1785	57.1734
8	7.2842	46.6786	-12.2274	36.2379	38.4231	47.0972
9	9.2419	45.9712	-12.3221	37.8227	39.2994	47.2921
10	8.6380	52.5742	-12.3697	39.9224	40.1153	54.5532
Average	9.3490	49.8001	-11.8909	38.4917	39.0629	51.6034

Fig. 5: EKF UAV Localization Simulation Results

Massachusetts Institute of Technology teamed up with NASA Langley Research Center to develop a multi-UAV system, aimed at search and rescue missions within a forest underneath the tree top canopies in a GPS denied environment. Several technical challenges were involved in their research. The first of which being that GPS is not available in most cases as it cannot penetrate the think forest canopies, thus as substitute, on board sensor estimation was required. Another obstacle was utilizing a collaborative mapping application that would incorporate map fusion to assist in UAV localization. Severe aliasing is common as there exist several similarities between groupings of trees. The multi-UAV search and rescue system's success was validated with both simulations and real-world exploration missions, relying on a ground station performing collaborative simultaneous localization and mapping (CSLAM) with support from an EKF based vehicle position state estimate. LIDAR based laser scans with a field of view of 270° locally create a compressed lightweight submap of the trees and obstacles within it close range environment. Later when communication is available the submaps are transmitted to the ground stations for CSLAM to be performed in two steps. The first step resolves the individual submaps from all the UAVs making up the system and the second step jointly optimizes the submaps together resulting in a global map. The EKF used to estimate six-degree-of-freedom pose was laser based and combined with measurements from a LIDAR sensor measuring outboard depth, a laser altimeter, and an IMU.

From the real-time EKF state estimates, a map of explored regions was created [23].

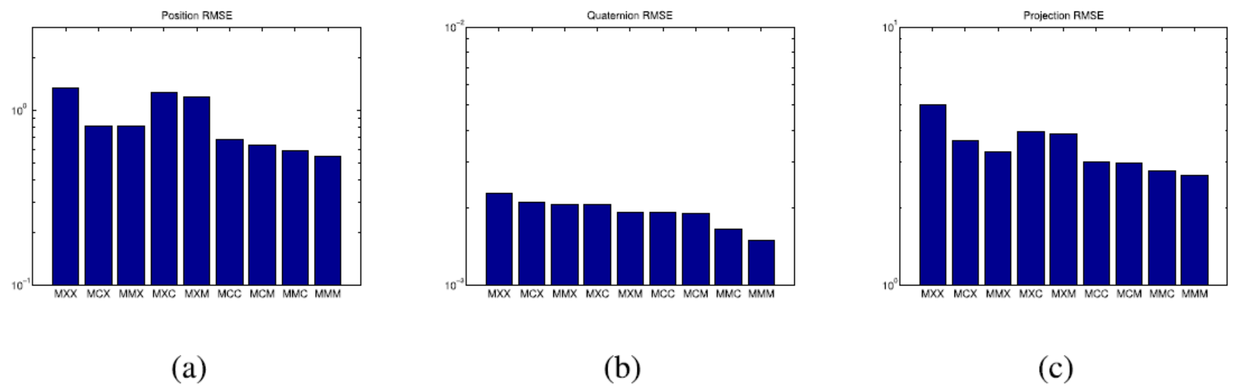


Fig. 6: Results from Real Data Comparisons for (a) Positions, (b) Orientation, and (c) RMSE

The framework of sensor fusion for tracking applications is further discussed in the conference paper, Fusion of Inertial and Vision Data for Accurate Tracking. Two approaches are investigated, a gyroscope only model for vision based tracking and an accelerometer (IMU) based model where both measurements from the accelerometer and the gyroscope along with vision data are used for estimating the camera's state (position, velocity, acceleration, and biases). Camera based tracking is only suitable for low frequency frequency state variations, whereas inertial sensors are ideal for high frequency motion tracking. Unfortunately IMU based measurements accumulate more and more noise through integration over time. The results concluded that the acceleration fusion model (Model 2) out performed Model 1. The acceleration fusion model used an EKF to integrate the different sensor measurements (gyroscope, accelerations, and visual sensors) in which the camera was assumed to maintain a constant angular velocity and acceleration. The model for the time update is given in Equation 27 in which q_t is the quaternion for the camera's attitude.

$$x_{t+\Delta t} = \begin{bmatrix} p_t + v_t\Delta T + 0.5a_t\Delta T^2 \\ v_t + a_t\Delta T \\ a_t \\ b_a \\ \begin{bmatrix} cos(0.5\omega_t\Delta T) \\ sin(0.5\omega_t\Delta T)\frac{\omega}{\|\omega_t\|} \end{bmatrix} \times q_t \\ \omega_t \\ b_\omega \end{bmatrix} \tag{27}$$

Simultaneous localization and mapping (SLAM) is again applied in a paper published in the Journal of Field Robotics to enable autonomous navigation in GPS denied environments. In this case the measurement sensors are an IMU as before and a 2D laser scanner. The 2D laser scanner is the input to a scan-matcher algorithm in which multiple scans are used to determine the likelihood of the Micro Air Vehicle (MAV) being on a know map representation and the state transformations between different points on the map making it possible to ultimate determine the distance between scans. When implementing a scan-matcher algorithm resolution is directly proportional to accuracy. For some autonomous ground robots a lower resolution is required as compared to a hovering autonomous vehicle. For a map resolution of 10cm, the noise associated is an RMS of about 0.5m/s, which can be removed with some basic low pass filtering, however this would induce a counter productive amount of lag. This is just one example of why sensor fusion is necessary. To control the MAV the position and velocity state estimates were achieved by fusing the scan-match change in position estimates with those of the IMU. The implementation had a maximum deviation from a straight path trajectory of 8cm. The state estimate were accurate enough to enable the MAV to fly and hover under the constrained GPS denied indoor environments [24].

2.4 GPS INTRUSION DETECTION SYSTEMS

As UAVs become more common across industries reaching from civil to defense applications so does the necessity for a stronger cyber defense. Some vulnerabilities include GPS Spoofing and Jamming. Jamming is typically the first step in a GPS Spoofing attack. GPS Jamming is the phrase used to describe an instance in which an outside source blocks/overwrites (Jams) the GPS signal being provided to the UAV. Once a GPS signal is Jammed, the flight controller is triggered and switches into auto mode, in which the autopilot takes over. This is proceeded by GPS Spoofing. During a GPS Spoofing misleading inputs are fed from the outside intruder to the flight controller which change a trajectory, modify the target location, or create a counterfeit home location (for return landing) [25] [26]. This obviously can play a devastating role in compromising missions and potentially even lead to casualties.

To combat such intrusions supervised machine learning algorithms can be implementing to develop a classification based intrusion detection system. Two prominent machine learning algorithms used in the literature are the Neural Network and Support Vector Machine. An in depth description of these algorithms is outside the scope of this paper therefore they will briefly be touched on as the application with regards to the subject of IDS is the main focus. In laymen terms a neural network consist of layers in which inputs are mapped to outputs. A multi-layer neural network consist of several hidden layers between the input and output in which several computations are performed. The single layer neural net only performs the computation during the output layer and is typically a binary process. The final stage of the neural network is the loss function at the output stage that is optimized with a softmax function for multi class (see Equation 28) or sigmoid function for a binary class (see Equation 29). This neural network is a supervised machine learning algorithm which requires training data with a known output. The weights for each input are optimized in the training potion of the classifier [27].

$$\Phi(v_1...v_k) = \frac{\begin{bmatrix} e^{v_1} & ... & e^{v_k} \end{bmatrix}}{\sum_{i=1}^{k} e^{v_i}} \tag{28}$$

$$\Phi(v) = \frac{1}{(1 + e^{-v})} \tag{29}$$

Support Vector Machine (SVM), also known as the Margin Classifier, is a supervised learning classification algorithm. It can operate with an infinite dimensional data set. SVM defines margins or boundaries between the data points in multidimensional space. The main goal of this algorithm is to find a flat boundary referred to as a hyper plane that leads to a homogeneous partition of the data. A good separation is achieved by the hyper plane that has the largest distance to the nearest training point. That hyper plane's distance or width is associated with the maximum margin. The data points from each class have at least one support vector. Figure 7 portrays a hyperplane discretely partitioning data.

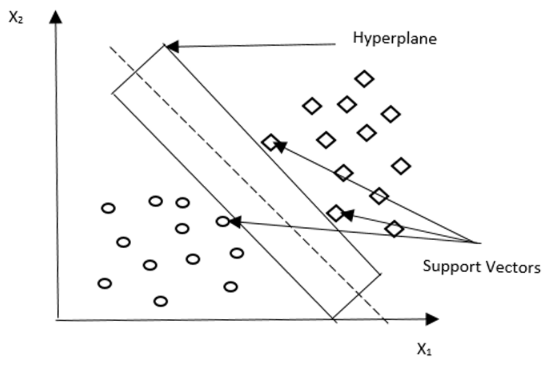


Fig. 7: Two Class Classification in SVM Linear Separable Case

If the data points are linearly separable then the hyperplane performs separation using convex hulls. The hyperplane is a perpendicular bisector of the shortest line between the two hulls. The equation of a hyperplane in n dimensions is given by,

$$\vec{w} * \vec{x} + b \geq 1$$

where w_1, w_2... w_n are weights and x_1, x_2...x_n are input features. The SVM algorithm finds the weights (W_n). Then, data points are separated accordingly.

$$\vec{w} * \vec{x} + b \geq 1$$
$$\vec{w} * \vec{x} + b < 1$$

Applications of vector geometry is used to define the distance between the two planes. In order to maximize the distance, $\|w\|^2$ must be minimized. A key feature of SVM is its ability to map the problem into a higher dimensional space using the Kernel method. Here, nonlinear space is transformed into linear space using a slack variable, a_j:

$$\frac{1}{2}||w||^2 + c\sum a_j \tag{30}$$

A cost parameter is added to the linear space. The objective is to try and minimize $c\sum a_j$, which is the upper bound on the number of misclassified data points. The Kernel Function is used to transform the data, x, from one space to another. It is a dot product of two functions, ϕ. Optimizations depends only on these dot products. The dot products of two vectors, K (x_i, x_j), has many kernel functions such as linear, polynomial, Gaussian, and Radial Basis Function (RBF) [2].

G. Panice et al. used Matlab's library for Support Vector Machines (LIBSVM) to develop a one class IDS. There was not access to data from a real GPS attack so a simulator was used for a preconfigured mission in which false GPS data was injected for the three case types, GPS Spoofing, GPS Jamming, and GPS Meaconing. Performance metrics for the tested IDS were false positive rate (FPR), true positive rate (TPR), and accuracy. The goal of the experimental IDS was to compare its implemented results in detecting a Spoofing attack to Receiver Autonomous Integrity Monitoring (RAIM), widely used in the aviation industry, and to evaluate the probability of failed detection. The proposed method had a similar performance to the more common RAIM, however the quality of the classifier degrades faster over time [7].

Another work explores the applications of a supervised neural network based INS with five main data features, satellite vehicle number (SVN), signal to noise ration (SNR), pseudo range (PR), Doppler shift (DO), and carrier phase shift (CP). GPS spoofing is achieved assuming a Meaconing attack. A Meaconing attack is receiving GPS signals in a remote location and the rebroadcasting the signal at another location at a slightly higher power [26] used to confuse navigation. Binary training and test data were collected by collecting GPS data at two know separate locations approximately 480 feet apart. One location was used as legitimate while the other was used as Spoofed (rebroadcasts). A series of Neural Networks were tested changing the computational algorithm, activation function, and the number of neurons in two hidden layers. The research concluded that an Adam-ReLu solver-activation pair had the greatest IDS accuracy and that the accuracy impact of adding additional neurons steadied out at 14 [26].

The final study considered herein was a comparison between an SVM IDS and a hybrid SVM IDS that relied on an unsupervised neural network for collecting training data (relevant GPS features). The self taught learning algorithm used as the solver in the deep neural network had three layers [25].

CHAPTER 3

METHODOLOGY

The main challenge of this research was to develop a navigation system not dependent on GPS or radio controlled communication such that it could maintain autonomous operations. Several key methods played a vital role in developing such a system. Fundamental principles relevant to Guidance Navigation and Control as well and the Extended Kalman Filter are discussed herein. Mavlink and dronekit based communication to the flight controller is also introduced.

3.1 STATE ESTIMATION

When dealing with the physical world one cannot yield an empirical description without some form of assumed causality. As such it is acceptable to assume a random phenomena as independent Gaussian processes. The Kalman Filter exist on the basis that a random function of time may be thought of as the output of a dynamic system excited by an independent Gaussian random process [13]. One of the most fundamental and well know probability distributions used throughout academia is the or Gaussian distribution. For continuous real variables, x, the Gaussian distribution is defined as:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} exp\{-\frac{1}{2\sigma^2}(x-\sigma)^2\} \tag{31}$$

The Gaussian distribution is dependent on two variables, the mean, μ, and the variance, σ^2. The distribution is normalized such that the area under the curve alway sums to 1, therefore the shape of the distribution changes based on its maximum occurring at the mean and the variance corresponding to the spread of values in the distribution. Figure 8 better portrays the influence of the mean and variance. For the application of the Kalman Filter and later the Extended Kalman Filter, it is ideal to think in terms of a dimensional vector of variables, like the ones coming from sensor measurements, and how they can be described by a Gaussian distribution.

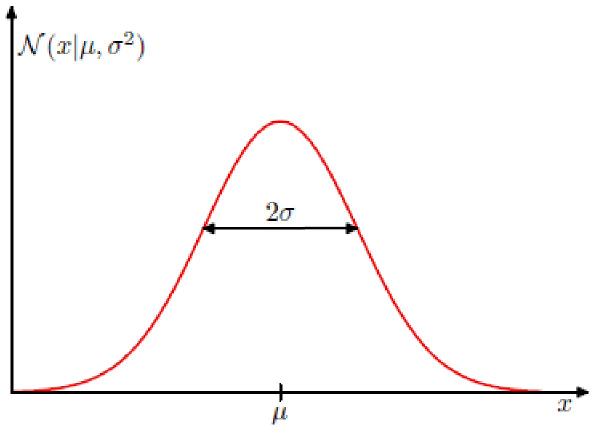


Fig. 8: [2] Univariate Gaussian Illustrating Dependencies on Mean and Standard Deviation ($\sqrt{\sigma^2}$)

An N-dimensional vector of x continuous variables has the following Gaussian distribution [2]:

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{32}$$

μ represents an N-dimensional vector of means where as Σ is the N x N covariance matrix. The diagonal of the covariance matrix is variance of the individual variables and the off diagonal elements are the covariances between the different variables. If the variable are independent of each other then their covariance goes to zero. A higher order Gaussian can be computationally costly as the determinant of the covariance matrix is a variable within the distribution ($|\Sigma|$).

The Bayes Filter lays the framework for many common recursive state estimators including the Kalman Filter and consequently the Extended Kalman Filter. It estimates the current state of a system given observations (measurements) and the input or control commands. The recursive nature of the algorithm allows the user to predict the state at time $t+1$ given the information at time t therefore it only requires the most recent system information. The Bayes Filter can be derived by applying the following: Bayes Rule, Markov assumption, and the law of total probability. Bayes Rule is a probabilistic theorem in which a decision about a future event is made from the past probabilities. In other words, a posterior distribution is made from evidence and a prior distribution. Bayes Rule is given below in Equation 33 where $k = 1, 2, 3...., k$.

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)} \tag{33}$$

$P(x|C_k)$ is a conditional distribution, $P(C_k)$ is the prior distribution and $P(x)$ is the normalizing evidence[2]. The Markov assumption is key when applying Bayes Rule in developing the Bayes Filter. It assumes the current state of the system to be decoupled from the past or future. Therefore, given the current state of the system, all previous control commands do not necessarily contribute to what will be observed in the present or future. It simplifies mathematical operations as it drops the past measurements and control inputs from the likelihood function of the new observation. The final tool necessary in deriving the Bayesian Filter is the law of total probability which allows one to partition the total probability into several smaller probabilities of the same sample space. The full derivation of the Bayesian Filter is given below in which the Markov assumption and application of the law of total probability are pointed out.

- applying Bayes Rule,

$$bel(x_t) = P(x_t|z_{1:t}, u_{1:t})$$

- applying the Markov assumption to the expression in red

$$bel(x_t) = \eta P(z_t|x_t, z_{1:t-1}, u_{1:t})P(x_t|z_{1:t-1}, u_{1:t})$$

- by law of total probability the expression is red becomes

$$bel(x_t) = \eta P(z_t|x_t)P(x_t|z_{1:t-1}, u_{1:t})$$

- applying the Markov assumption to the expression in red

$$bel(x_t) = \eta P(z_t|x_t) \int P(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})P(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1}$$

- applying the independence assumption to the expression in red

$$bel(x_t) = \eta P(z_t|x_t) \int P(x_t|x_{t-1}, u_t)P(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1}$$

- simplifying to

$$bel(x_t) = \eta P(z_t|x_t) \int P(x_t|x_{t-1}, u_t)P(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1}$$

$$bel(x_t) = \eta P(z_t|x_t) \int P(x_t|x_{t-1}, u_{1:t})bel(x_{t-1})dx_{t-1}$$

The final expression of the Bayes Filter consist of two steps: the prediction step and the correction step. The prediction step is depends on the motion model to advance the state based on the command. The correction step consists of an observation or measurement model describing the likelihood of the observations, given the state is known. At each time, t, a prediction and correction is being made. Equations 34 and 35 illustrate the prediction step with the motion model underlined and the correction step with the observation underlined accordingly.

$$b\bar{e}l(x_t) = \int \underline{p(x_t|u_t, x_{t-1})} bel(x_{t-1}) dx_{t-1} \tag{34}$$

$$bel(x_t) = \eta \underline{p(z_t|x_t)} b\bar{e}l(x_t) \tag{35}$$

As the Bayes Filter only lays the frameworks for recursive state estimators, various filters can be derived based on the application of different realizations and assumptions. The Extended Kalman Filter introduced in the previous section is a modified Bayes Filter where probability distributions are realized as Gaussian distributions and the Taylor series approximation is used to linearize non-linear models.

3.2 TAYLOR SERIES EXPANSION

For linearization over a small range $(x - x_0)$ Taylor series expansion is a suitable operator. Many mechanical systems have nonlinear components and to arrive at more manageable transfer function linear approximations of the nonlinear system must be obtained. For a nonlinear differential Equation linearization must occur for small-signal inputs about a steady or equilibrium state [3]. The smaller excursions from the point about which one is applying the linearization allows the higher order terms in a Taylor series expansion to be neglected. The full expression for a Taylor series expansion is given in Equation 36.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ... + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{36}$$

For small excursions, Equation 36 reduces to Equation 37 better portrayed in Figure 9 where $\delta f(x)/\delta x$ is the instantaneous slope, $\frac{df}{dx}$.

$$f(x) = f(x_0) + \frac{df}{dx}|_{x=x_0}(x - x_0) \tag{37}$$

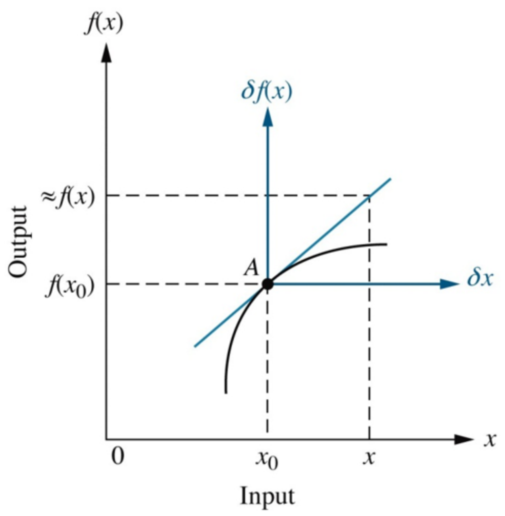


Fig. 9: Linearizion about Point A [3]

3.3 QUADROTOR DYNAMIC MODEL

The Quadrotor used for the EKF application is best described as a 6 degrees of freedom (DOF) system. It is based on two different reference frames, the inertial frame and body frame. The inertial frame or the frame with a fixed coordinate system was one in which Newton's laws are all considered to be true. The cardinal axis of the inertial frame are north, east, and down (towards the center of the earth). The body frame is referenced to the body center of the vehicle. Simply put, the inertial frame is earth centric, where as the body frame is vehicle centric. Typical forces and moments driving the Quadrotor state space system is composed of the thrust generated from the rotor rotation, pitching moments, and rolling moments. The six degrees of freedom system is based on the translational and rotational motion along/about the three body frame axis. The control of the 6 DOF system in achieved by modifying the rotational speed of the four motors. The motion of the vehicle can be described by a mix of reference coordinates. For example translational kinematic equations can be derived in the inertial frame, whereas translational kinetic equations, and rotational kinetic and kinematic equations can be derived in the body frame. When dealing with models dependent on mixed coordinate systems, transformations from one frame of reference (FOR) to the next are essential.

Euler angles are angles used to describe the orientation of a rigid body in 3D Euclidean space. For the purposes of this research the Euler rotations will be represented by rotations about Z, Y, and X consecutively. Euler angles can be used to transform the coordinates of a point in one FOR to another FOR, for example an inertial frame to a body frame. Euler angles represent a sequence of three elemental rotations about an axes of a coordinate system. Any orientation can be achieve in such an operation. The following rotations about the Z, Y, and X axes (yaw (Ψ), pitch (Θ), roll (Φ)) describe the three Euler rotations applied to go from an inertial FOR to a body FOR.

$$R_z(\psi) = \begin{vmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{vmatrix} \tag{38}$$

$$R_y(\theta) = \begin{vmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & \cos(\theta) \end{vmatrix} \tag{39}$$

$$R_x(\phi) = \begin{vmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{vmatrix} \tag{40}$$

The product of the three transformations can be used to go directly from one frame to the other. The inverse is also true. The transformation from the inertial frame to the body frame is given below ($T = R_x(\phi) \cdot R_y(\theta) \cdot R_z(\psi)$) where C represents a cosine function and S represents a sine function.

$$T(\psi, \theta, \phi) = \begin{vmatrix} C(\psi)C(\theta) & S(\psi)C(\theta) & -S(\theta) \\ C(\psi)S(\theta)S(\phi) - S(\psi)C(\phi) & S(\psi)S(\theta)S(\phi) + C(\psi)C(\phi) & C(\theta)S(\phi) \\ C(\psi)S(\theta)C(\phi) + S(\psi)S(\phi) & S(\psi)S(\theta)C(\phi) - C(\psi)S(\phi) & C(\theta)C(\phi) \end{vmatrix} \tag{41}$$

3.3.1 EKF MMM

The EKF approach MMM corresponds to a state space system used for the Quadrotor in which there is no control input and both acceleration sensor data and gyroscope sensor data are used as measurements of the state. The process equations used to describe the dynamic equations of motion is given below in Equation 42 with the following state vector:

$$x = \begin{bmatrix} s \\ v \\ a \\ \delta \\ \omega \end{bmatrix}$$

$$\begin{aligned} s_t &= s_{t-1} + Tv_{t-1} + \frac{1}{2}T^2 a_{t-1} \\ v_t &= v_{t-1} + Ta_{t-1} \\ a_t &= a_{t-1} \\ \delta_t &= T\omega_{t-1} \\ \omega_t &= \omega_{t-1} \end{aligned} \tag{42}$$

s, v, and a represent the translational motion (position, velocity, and acceleration respectively) in the x, y, and z directions and δ and ω represent the rotational motion (attitude and velocity) about the x, y, and z axis. Calculations are made at each time step, T, to determine the state which is filtered through the application of the EKF algorithm described in detail in Section 2.3. For each loop, a measurement update is made followed by the calculation of the Kalman gain after which the state estimates are calculated along with an update to the error covariance. The loop finishes with a new projection of the estimates in the form of a predicted new state.

The following Jacobian matrices were used in the EKF algorithm as described in Section 2.3. The W and V Jacobian matrices consist of partial derivatives with respect to process and measurement noise. For W the partial derivatives are of the function relating the state at time step t to the state at time step $t+1$ and for V the partials are of the function relating the state at time t to the measurements at time t. For this case, there is not a structured noise as part of our process or measurement equations and therefore they are treated as identity matrices. The priori and posteriori covariances are initialized as identity matrices as well before being updated at each time step of the computation.

$$A = \begin{pmatrix}
1 & 0 & 0 & dt(i) & 0 & 0 & \frac{dt(i)^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & dt(i) & 0 & 0 & \frac{dt(i)^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & dt(i) & 0 & 0 & \frac{dt(i)^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & dt(i) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & dt(i) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt(i) & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & dt(i) & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & dt(i) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & dt(i) \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

$$H = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

3.4 NONLINEAR OPTIMIZATION

Nonlinear optimization has a role to play across the spectrum of industry and research. A well known nonlinear optimizer is simulated annealing. The simulated annealing algorithm is based on the process of cooling molten hot materials at a slow and steady rate such that it yields a stable state with all the molecules aligned. The benefit of the algorithm is that it uses a wide spread of inputs to characterize the state while reducing a cost function. A relationship between cost and temperature is made such that as temperature goes down the algorithm weights the cost higher and as it increases the algorithm weights the costs less. Essentially the algorithm chooses more random inputs to solve for a global minima at the start of execution when temperature is high. As time increases and temperature slowly decreases the algorithm chooses inputs over a smaller range basing decisions more heavily on the cost. Equation 43 displays the probability of acceptance used for systematically lowering the system's temperature and storing the best points found for a given interval (points that reduce the cost function) [28].

$$\frac{1}{1 + exp(\frac{\Delta}{max(T)})} \tag{43}$$

3.5 QUADCOPTER COMMUNICATION PROTOCOLS

Dronekit is an open source, python based, application program interface (API). It allows for the development of additional application that run on a companion computer communicating with the ArduPilot flight controller. It increases the functional autonomous flight capability of the drone. The Quadcopter developed for this research relied on Dronekit to execute autonomous flight missions. The API establishes communication between the companion computer and the flight controller over Mavlink based communication protocols. With the help of Mavlink, access to a connected vehicles' state and parameters is realized. The Dronekit API provides methods to achieve the following [29]:

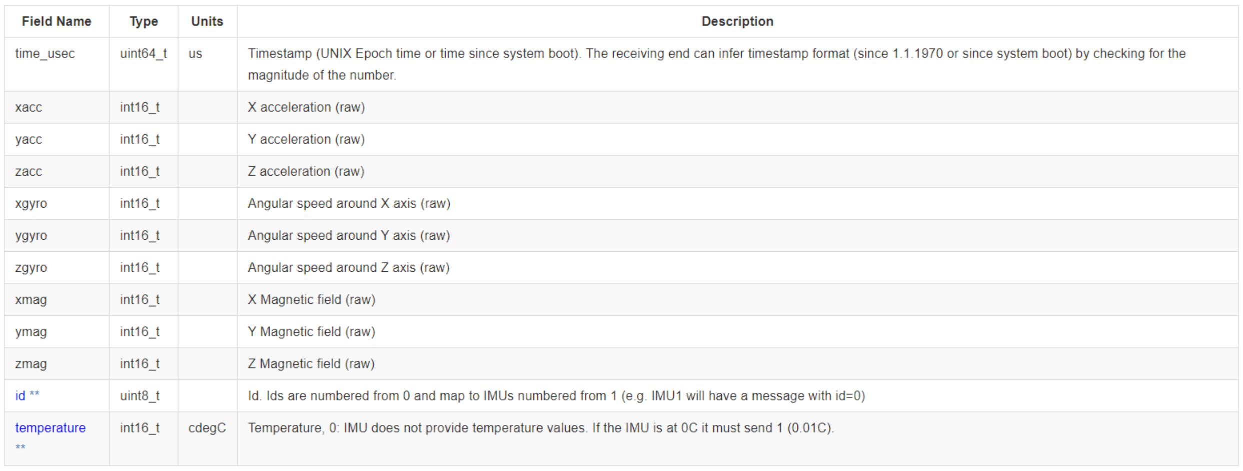
- Connect to a vehicle (or multiple vehicles) from a script
- Get and set vehicle state/telemetry and parameter information.
- Receive asynchronous notification of state changes.
- Guide a UAV to a specified position (GUIDED mode).

- Send arbitrary custom messages to control UAV movement and other hardware (GUIDED mode).
- Create and manage waypoint missions (AUTO mode).
- Override RC channel settings.

Dronekit can run on multiple operating systems, the most common being, Windows, Mac OS X, and Linux. Dronekit is used as a central database residing on the Quadcopters companion computer giving the drone operator access to the same data features accessible to the flight controller. Dronekit was also used for software in the loop (SITL) simulations prior to the Quadcopter development or test flights. This made it possible to debug the scripts using Mavlink messages to command the Cube Black autopilot.

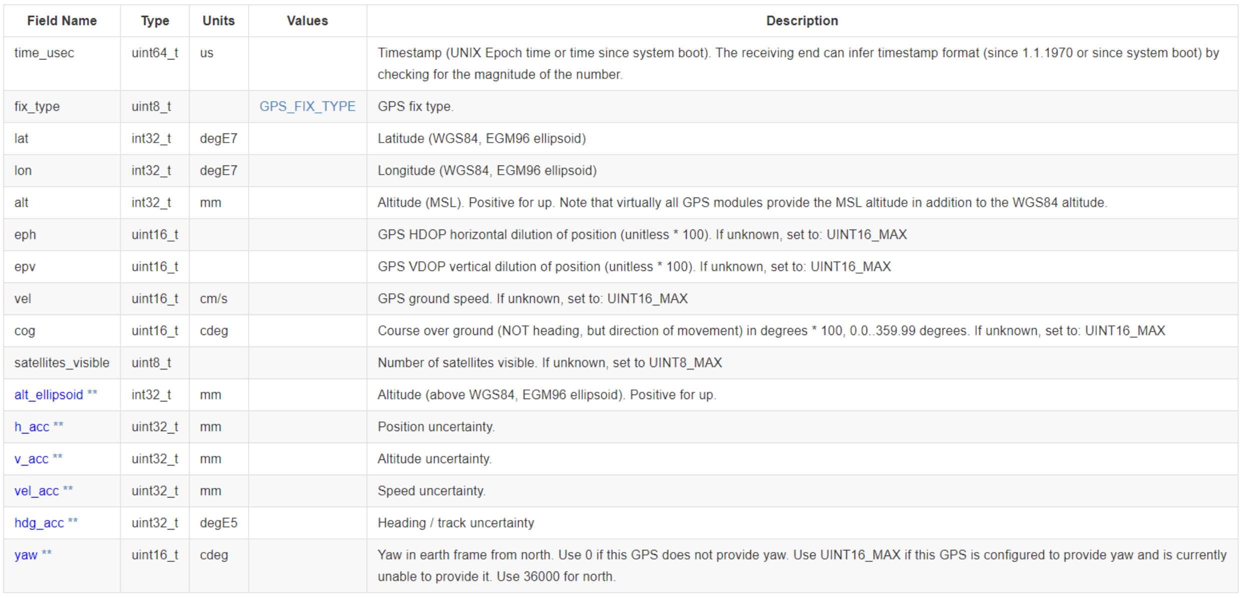
Mavlink is a lightweight communication protocol used for messaging data between various components on a drone. Mavlink publishes key data as messages or topics from a drone and can be broadcast to subsystems such as other drones or ground control stations. Their are many different code generated software libraries that work with the existing XML Mavlink files (Python based Dronekit for example). MAVLink is an efficient and reliable on board and off board communication protocol that can be used with many different languages (C, C++, C#, Python, Java Script, etc.). It can allow connectivity between up to 255 concurrent systems. It is a "binary based telemetry protocol for resource constrained systems and bandwidth constrained links" [30]. Dronekit uses Pymavlink as a framework to process Mavlink messages being sent and received from the autopilot. In general there are two types of Mavlink messages those being sent from the companion computer to the flight controller and the second being the messages being received by the companion computer.

The first type of messages are used to command settings such as flight mode and the vehicles airspeed, position, or altitude. The encoding structure is either "COMMAND_INT" or "COMMAND_LONG". "COMMAND_INT" is applicable when the coordinate reference frame is relevant such as when setting waypoints to fly to autonomously. "COMMAND_LONG" is embedded in Dronekit commands that set certain parameters of the vehicle like airspeed or altitude. The second type of message is used to collect certain attributes in a data stream. For this research the two most important are the "RAW_IMU" messages and the "GPS_RAW_INT". These message descriptions are illustrated in Figures 10 and 11. These output the raw measurements recorded by the onboard GPS sensor and the IMU internal to the Cube Black autopilot. A full listing of the available MAVLink messages can be found in the online documentation available at: https://mavlink.io/en/messages/common.html.



Field Name	Type	Units	Description
time_usec	uint64_t	us	Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude of the number.
xacc	int16_t		X acceleration (raw)
yacc	int16_t		Y acceleration (raw)
zacc	int16_t		Z acceleration (raw)
xgyro	int16_t		Angular speed around X axis (raw)
ygyro	int16_t		Angular speed around Y axis (raw)
zgyro	int16_t		Angular speed around Z axis (raw)
xmag	int16_t		X Magnetic field (raw)
ymag	int16_t		Y Magnetic field (raw)
zmag	int16_t		Z Magnetic field (raw)
id **	uint8_t		Id. Ids are numbered from 0 and map to IMUs numbered from 1 (e.g. IMU1 will have a message with id=0)
temperature **	int16_t	cdegC	Temperature, 0: IMU does not provide temperature values. If the IMU is at 0C it must send 1 (0.01C).

Fig. 10: The RAW IMU Readings For a 9DOF Sensor, Which Is Identified by The ID (Default IMU1) [4].



Field Name	Type	Units	Values	Description
time_usec	uint64_t	us		Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude of the number.
fix_type	uint8_t		GPS_FIX_TYPE	GPS fix type.
lat	int32_t	degE7		Latitude (WGS84, EGM96 ellipsoid)
lon	int32_t	degE7		Longitude (WGS84, EGM96 ellipsoid)
alt	int32_t	mm		Altitude (MSL). Positive for up. Note that virtually all GPS modules provide the MSL altitude in addition to the WGS84 altitude.
eph	uint16_t			GPS HDOP horizontal dilution of position (unitless * 100). If unknown, set to: UINT16_MAX
epv	uint16_t			GPS VDOP vertical dilution of position (unitless * 100). If unknown, set to: UINT16_MAX
vel	uint16_t	cm/s		GPS ground speed. If unknown, set to: UINT16_MAX
cog	uint16_t	cdeg		Course over ground (NOT heading, but direction of movement) in degrees * 100, 0.0..359.99 degrees. If unknown, set to: UINT16_MAX
satellites_visible	uint8_t			Number of satellites visible. If unknown, set to UINT8_MAX
alt_ellipsoid **	int32_t	mm		Altitude (above WGS84, EGM96 ellipsoid). Positive for up.
h_acc **	uint32_t	mm		Position uncertainty.
v_acc **	uint32_t	mm		Altitude uncertainty.
vel_acc **	uint32_t	mm		Speed uncertainty.
hdg_acc **	uint32_t	degE5		Heading / track uncertainty
yaw **	uint16_t	cdeg		Yaw in earth frame from north. Use 0 if this GPS does not provide yaw. Use UINT16_MAX if this GPS is configured to provide yaw and is currently unable to provide it. Use 36000 for north.

Fig. 11: The Global Position, as Returned by The Global Positioning System (GPS) [4].

CHAPTER 4

EXPERIMENTAL SETUP

The experimental development of the Quadcopter proceeded in accordance with the research objectives. It was broken into two parts. The first part consisted of generating Dronekit based Python scripts supporting autonomous flight and verifying their performance. This step was critical as it served as a proof of concept to whether or not autonomous mission were obtainable. Not only was code required for programmable autonomous missions but, it was also required to record relevant onboard sensor data (achievable via Mavlink messaging). The entirety of the preliminary design stage (Part I) would not have been possible without the application of software in the loop (SITL) simulations. Part II of the Quadcopter design focused on hardware and assembly. This includes every component used for a baseline functioning drone and all of the additional high end sensors and companion machines required for the EKF measurements. The following sections will dive deeper into the intricacies of the experimental setup obligatory for Part I and Part II.

4.1 SOFTWARE

A suite of software packages was required to complete a thorough simulation to checkout the developed autonomous flight programs. For simplicity everything was performed on a Linux based virtual machine (VM), Ubuntu 20.04. The following subsections note the key software packages and developed python scripts to support the software in the the loop simulations along with a brief description of each.

4.1.1 ARDUPILOT

ArduPilot is an open source autopilot firmware that is used across several platforms to include multi-copters, helicopters, fixed wing aircraft, submarines, and rovers. The main flight code is written in C++. Several flight controllers are supported with ArduPilot firmware such as, Pixhawk, The Cube, Pixracer, NAVIO2, and Bebop2. More information regarding ArduPilot can be found using Reference [31].

4.1.2 SITL

SITL is a software in the loop simulator that allows the user to simulate various ArduPilot vehicle platforms including, ArduPlane, ArduCopter, ArduSub, and ArduRover. As the developed UAV was a Quadcopter, the ArduCopter simulation package was used. The platform is a copy of the autopilot code using an common C++ compiler. The platform allows to execute and test user defined scripts without requiring any hardware in the loop. When executing a simulation using SITL, the sensor data from the drone are artificially fed from the flight dynamics model associated with the user specified platform (ArduCopter in this case). SITL is advantageous as it gives developers access to a wide range of coding tools to assist in static and dynamic analyzers, debugging tools, and the ability to easily implement code changes to test rapidly. Figure 12 illustrates the main architecture of the simulated operation. References [32] & [33] were used to set up and get started with SITL on the Ubuntu virtual machine.

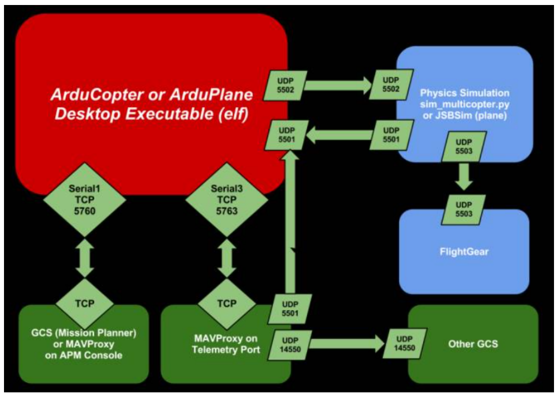

ArduCopter or ArduPlane Desktop Executable (elf)
UDP 5502
UDP 5502
Physics Simulation sim_multicopter.py or JSBSim (plane)
UDP 5501
UDP 5501
UDP 5503
UDP 5503
FlightGear
Serial1 TCP 5760
Serial3 TCP 5763
TCP
TCP
GCS (Mission Planner) or MAVProxy on APM Console
MAVProxy on Telemetry Port
UDP 5501
UDP 14550
UDP 14550
Other GCS

Fig. 12: Overview of SITL Simulation Architecture [5]

4.1.3 QGROUNDCONTROL

QGroundControl is a software based Graphical User Interface (GUI) supported across multiple operating systems such as Linux, Max iOS X, Android, and windows. It is typically run as the ground control station for PX4 or ArduPilot supported vehicles. For SITL simulations it was run on the Ubuntu virtual machine. For the purposes of this research is was used during simulations as a setup and configuration interface of the ArduCopter vehicle. From QGroundControl some nominal autonomous missions may be set by the operator but, more importantly a flight map is displayed along with vehicle position/flight tracking, waypoints, and instrumentation outputs. The software is supported by Mavlink communication protocol between the ground control station and the flight controller [34]. Figure 13 portrays an instance QGroundControl was used during the SITL simulations run on the VM.

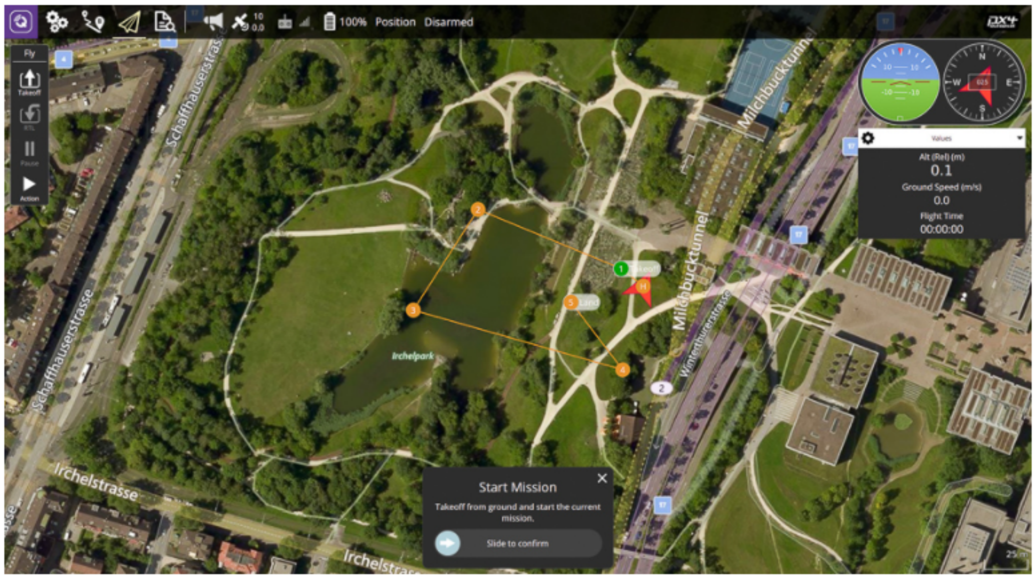


Fig. 13: Basic QGroundControl Interface for SITL

4.1.4 AUTONOMOUS MISSION SCRIPTS

The codes intended to be run supporting autonomous missions during the flight test stage of this research were vetted in SITL ArduCopter simulations. To gain familiarity with the Mavlink supported Python libraries, several examples were run and are available for reference. Three main codes were developed in support of the EKF development broken down as follows: An autonomous mission script aimed at flying to various waypoints at a given altitude and velocity, a take data script recording the nominal readings from onboard sensors used in the autopilot controls, and a Spoof GPS script sending Mavlink messages to the autopilot feeding the autopilot artificial user defined GPS data.

- Fly_Mission.py: The autonomous flight mission connects to the flight computer via a connection string. Once the connection is verified the motors become armed and the vehicle mode transitions to "GUIDED" and takes off to a desired altitude of 40 meters and the vehicle airspeed is set to 3 meters per second. The mission then begins flying to four separate waypoints defined by their GPS coordinates. For more details, see Appendix A.

- Take_Data.py: The python script that runs during the flight test to record sensor outputs relies on several native python libraries like numpy and pandas for matrix operations. The routine first creates Mavlink attributes so that the sensor outputs may be called throughout the code. The GPS and IMU sensor outputs are arranged in a labeled table and exported as a .csv file compatible with Microsoft excel and Matlab. For more details, see Appendix A.

- GPS_Spoof.py: The GPS position predicted from the EKF utilizing the external IMU measurements was relayed to the onboard flight controller with Pymavlink functions. The Mavlink functions created raw packages of messages to be sent and abstracted. The command associated with sending GPS information to be used with the autopilot is "mav.gps_input_send(...)". For more details, see Appendix A.

4.1.5 MISSION PLANNER

For flight tests the ground control station operated the most recent version of Mission Planner. Mission planner shares similar features with QGroundControl (used during the simulation stage of this research). The configuration of the Quadcopter was set in mission planner. It was also used to complete the required calibration necessary to to arm the Quadcopter's four motors. In order to interface the UAV with the ground control station a telemetry radio is connected to the machine hosting Mission Planner (via USB) and a baud rate of 57600 on COM port 7 is set.

4.2 HARDWARE

The Quadcoptor used for the purposes of this research was originally constructed from a HEXSOON EDU450 kit. This kit included the UAV's frame, four T-MOTOR KV880 brushless motors, a power distributor, and four propellers. The remaining components of the build were additional sensors, a flight controller, a battery, a telemetry radio, a transmitter, a serial to USB converter, a companion computer, a Teensy board, and an external IMU. An itemized list of the flight hardware is given in Table 1 including a brief description of critical components of the Quadcopter. Once the fundamental construction of the drone was complete the additional hardware was incorporated. The onboard telemetry radio was required to communicate with the ground control station software passing along the necessary telemetry data to ensure proper tracking. Figure 14 illustrates the UAV build described above that executed the autonomous mission scripts to perform the flight test missions. The figure identifies some of the key hardware called out in Table 1. Another drone was also used to execute some of the flight test missions via Mission planner. This was for simplicity once autonomous flight capability was achieved with the designed UAV for this research. The Teensy board responsible for data acquisition of the ADIS 16475 is shown mounted to the rigid body in Figure 15.

To run the autonomous flight scripts to perform the flight test missions and record data a raspberry pi was used as a companion computer. The use of a companion computer ensured the processing requirements of the Mavlink based codes did not hinder the performance of processor built into the Cube Black autopilot. As a work-around to communication issues via the serial port on the Pixhawk flight controller preventing a reliable connection with the raspberry pi (RPI), a serial to USB converter was implemented. This provided an interface between a serial port and a USB 2.0 port on the RPI. The Teensy board providing the data acquisition from the external IMU was powered via the Pixhawk and mounted to the Quadcopter's frame. The Teensy was loaded with C++ code establishing the capability of reading from the sensor's data registers and writing them to a micro SD card built into the board, thus allowing for real time acquisition. The pinouts of the IMU, Teensy 3.6 board, and RPI can be found in Appendix D.

TABLE 1: Breakdown of Quadcopter Hardware

Item	*Part Number*	*Manufacturer*	*Description*
1	HX4-06005	Hex Technology	ArduPilot Flight controller autopilot
2	RASPBERRY PI 3 MODEL B+	Raspberry Pi	Computer featuring Broadcom BCM2837B0, 64-bit SoC @ 1.4 GHz, 1 GB LPDDR2 SSDRAM, and 4 USB 2.0 ports
3	AIR2216 KV880	T-MOTOR	BLDC outrunner motor (brushless)
4	14057	Hex Technology	High precision GNNS GPS module supporting RTK mode and supports CAN BUS protocol
5	GEA50004S45D	Gens ACE	Four cell LiPo battery, 14.8V-74Wh, 5000mAh
6	M10013-RK	mRobotics.io	mRo Sik Telemetry Radio V2 915Mhz
7	SPM9645	SPEKTRUM	DSMX remote receiver operating on a 2.4GHz band
8	DEV-09873	SparkFun	Serial to USB converter, FTDI basic breakout - 3.3V
9	Teensy 3.6	PJRC	32 Bit Arduino-Compatible Microcontroller used to interface with the Analog devices IMU allowing for real time data sampling
10	ADIS 16475	Analog Devices	6DOF 12 output channel inertial measurement unit and breakout board

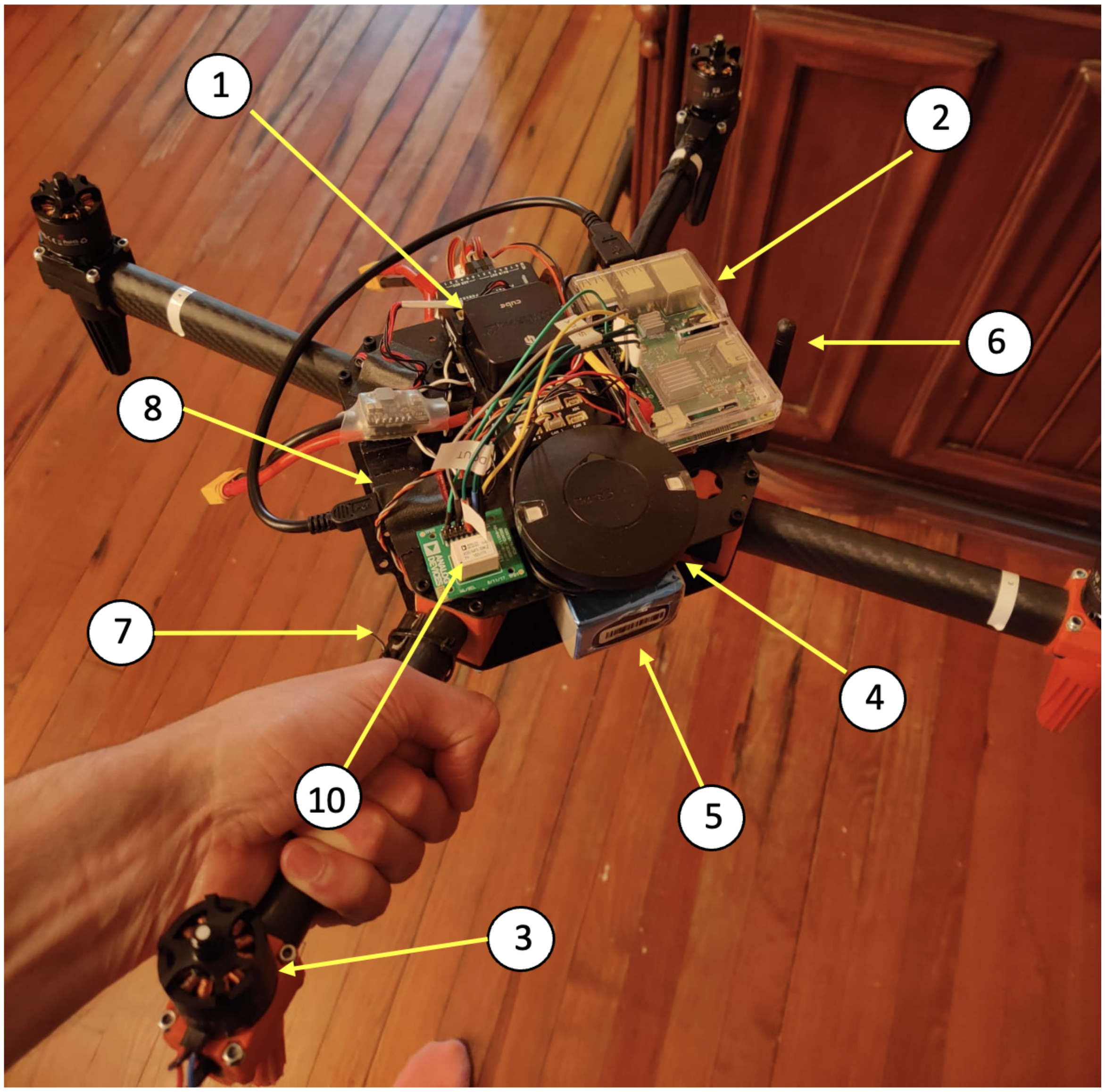


Fig. 14: Identified Quadcopter Hardware Components

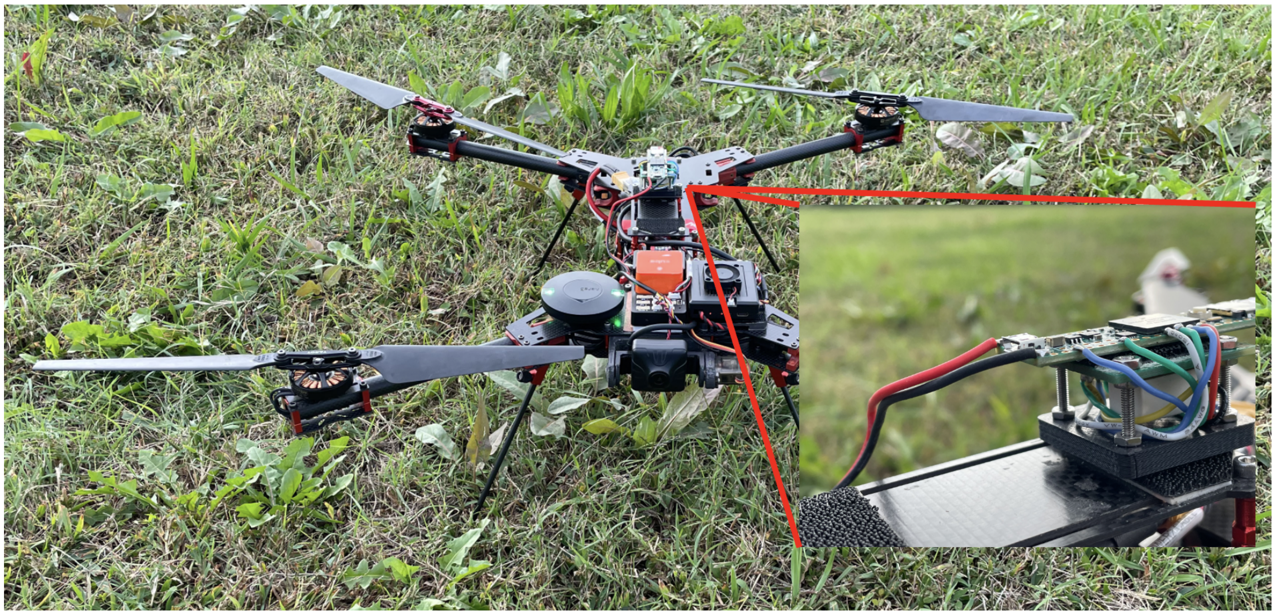


Fig. 15: Additional Quadcopter with Teesny IMU Interface Blown Up

Some of the components of the drone that were more essential for fundamental flight capability are the motors, battery, and receiver. The four electric motors used to provided the thrust necessary for flight were three phase motors associated with an RPM of 880 per 1 volt applied (under no load). There are twelve wound poles in the stator and fourteen magnets on the rotor (Figure 17). The thrust is generated from the induced torque resulting from the interaction of magnetic fields between the rotor and stator. The three phase design maximizes the intensity of the combined magnetic field associated with the alternating current (AC). The three sets of coils are offset by 120 degrees (Figure 16), keeping the power provided optimized. The motor's electronic speed controller (ESC) has a signal refresh rate of up to 600Hz and a range of 1ms to 2ms corresponding to 0% and 100% throttle.
The battery has four cells in series and one in parallel in which each cell has a nominal voltage of 3.7V. The battery cells cannot go below a 3.0V discharge as it damages the cell's chemistry and it is safest to maintain a full discharge threshold of 3.2V. A voltage of 16.8V was considered a full charge for Quadcopter flight test operations and 13V was considered a drained battery no longer capable of maintaining safe operation. The maximum continuous current draw of the LiPo battery was 225 Amps. The last component to mention is the

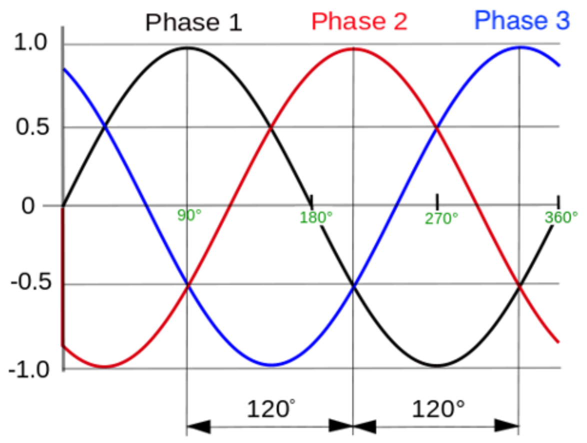


Fig. 16: Idealized Phase Diagram for a Three Phase Induction Motor

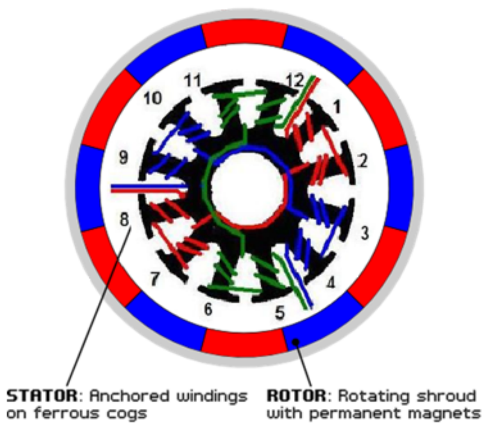

1
2
3
4
5
6
7
8
9
10
11
12
STATOR: Anchored windings on ferrous cogs
ROTOR: Rotating shroud with permanent magnets

Fig. 17: Electric Motor Stator Configuration

receiver located on one of the arms of the UAV. It was essential for manual operation of the Quadcopter during flight ensuring that the hand held controller inputs were communicated with the flight controller.

4.2.1 ADIS 16475

Analog Devices' ADIS 16475 is a precise micro-electromechanical system (MEMS) IMU consisting of 6 DOF sensors, being a triaxial accelerometer and a triaxial gyroscope. The manufacturers fused each sensor with signal conditioning to optimize the transducers dynamic performance. To ensure the most accurate measurement is provided, an extensive factory calibration is applied encompassing the characterization of each sensor's bias, alignment, sensitivity, and point of percussion. The IMU serves as a cost effective method for providing multiaxis inertial sensing for the industry with an easy to use serial peripheral interface (SPI) register structure for collecting data. The sensor is approximately 11mm x 15mm x 11mm on a break out board supplying the pins required for sampling connectivity. Figure 18 illustrates the functional block diagram of the ADIS 16475.

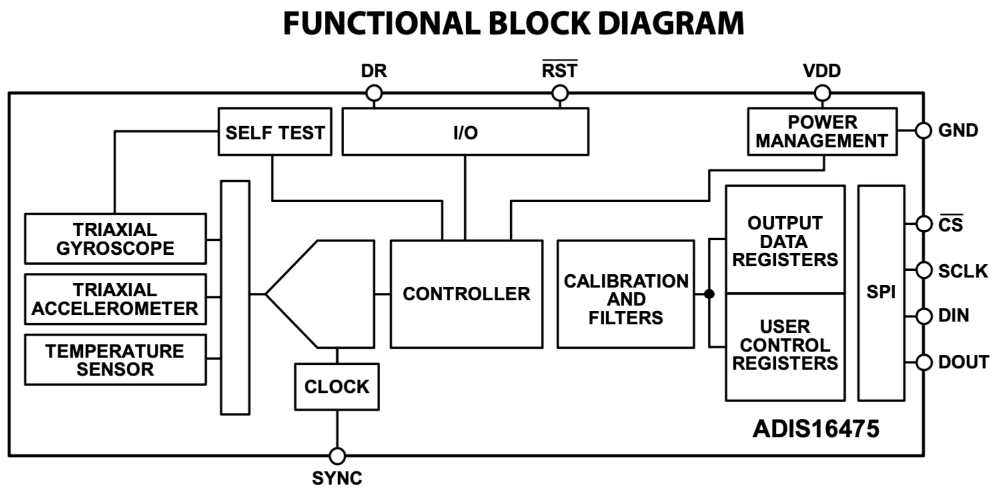


Fig. 18: Analog Devices ADIS 16475 Data Sheet Block Diagram

CHAPTER 5

FLIGHT TEST

After all the preliminary development of the autonomous mission was completed using SITL, and the Quadcopter build was finished successfully with the necessary hardware integrated into the system, a series of flight tests were required. The flight tests were critical in collecting IMU data during an autonomous mission to test the hypothesis of whether or not strapdown navigation relying on the Analog devices IMU alone would be a reliable solution for instances of flight in a GPS denied environment. Such environments could be associated with a GPS blackout, GPS jamming, or GPS spoofing. The flight tests consisted of many steps. The most significant aspects of which will be explained within this Chapter.

5.1 CONNECTING TO THE DRONE

Two different connection methods were used for the series of flight tests. If performing the dronekit based autonomous missions developed using SITL, a connection string via Mavlink was required to initialize the mission. First, an SSH connection had to be made between the ground control station and the onboard companion computer (RPI). Next, a serial to USB converter was used to provide this connection as mentioned in Section 4.2. The script was executed on the companion computer (RPI), passed to the USB to serial converter, and then to the serial port on the Pixhawk. ”Connect(’–connect’, default=“/dev/ttyUSB0”, wait_ready = True, baud=115200)” is the physical code initializing the connection between the drones’ flight controller and the user defined mission on the RPI at a baud rate of 115200.

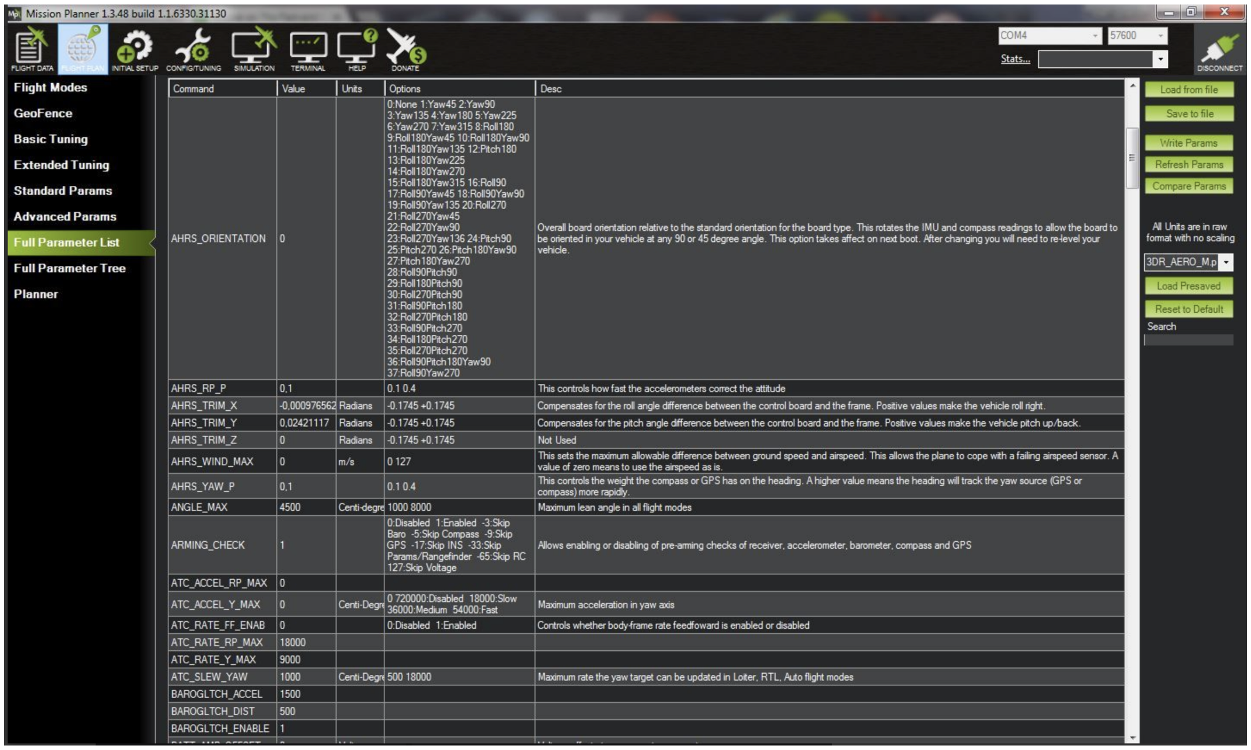
The second method of running the autonomous mission was done relying on an external software, Mission planner. The connection between the Quadcopter and the ground control station was made via a telemetry radio. Within Mission Planner a baud rate of 57600 had to be selected and set for the connection via the telemetry radio to be established. For both methods of connection certain settings of the flight controller must be enabled to ensure a successful and safe flight.

5.2 SENSOR INTERFERENCE AND FLIGHT SAFETY

Various parameters can be configured within Mission Planner or QGroundControl to ensure an autonomous mission cannot be completed without meeting certain thresholds of paramount importance ensuring a safe flight. Some of the safety related parameters to mention are that of battery level, GPS HDOP reading, and the number of GPS satellites available. A battery monitor exists within Mission Planner constantly checking the voltage and current of the battery. A return to launch (RTL) is set to be enabled at a user specified voltage level to avoid the possibility of an end of flight scenario due to the battery dying mid flight. The HDOP reading is a check of the GPS measured position accuracy. An HDOP value of under 2.0 is acceptable for a safe flight. The user cannot arm the drone (unless it is in a flight mode that doesn't require GPS) without meeting the appropriate HDOP level. As the autonomous mission requires flying the Quadcopter with the GPS sensor in the loop for navigation (in GUIDED mode) a 3D lock confirmation is required. A minimum of 6 GPS satellites are necessary to enable GUIDED flight mode, ensuring a safe flight.

During the initial checkout of the drone build some configuration restrictions were realized to avoid sensor interference. The most significant of which was the placement of the external Analog Devices IMU with respect to the HERE3 GPS sensor. GPS sensors are inherently sensitive to electrical noise and having the IMU placed in the vicinity of the GPS sensor resulted in an unacceptable HDOP level. This prevented the drone from arming in GUIDED mode. Simply moving the IMU further away from the GPS sensor remedied this issue.

Every configurable setting is available in the configuration tab of Mission Planner under the "Full Parameter List". This interface is where the baud rate is set, the telemetry port configuration is specified (allowing for a specific COM port to be accessible to the companion computer), ABSD-OUT settings are disabled, and the default data logging trigger is defined. These are the most noteworthy user specific parameters essential for this research but, several more capabilities and functions are configurable here. Figure 19 portrays the user interface given in Mission Planner when accessing the mentioned fields.



Command	Value	Units	Options	Desc
AHRS_ORIENTATION	0		0:None 1:Yaw45 2:Yaw90 3:Yaw135 4:Yaw180 5:Yaw225 6:Yaw270 7:Yaw315 8:Roll180 9:Roll180Yaw45 10:Roll180Yaw90 11:Roll180Yaw135 12:Pitch180 13:Roll180Yaw225 14:Roll180Yaw270 15:Roll180Yaw315 16:Roll90 17:Roll90Yaw45 18:Roll90Yaw90 19:Roll90Yaw135 20:Roll270 21:Roll270Yaw45 22:Roll270Yaw90 23:Roll270Yaw136 24:Pitch90 25:Pitch270 26:Pitch180Yaw90 27:Pitch180Yaw270 28:Roll90Pitch90 29:Roll180Pitch90 30:Roll270Pitch90 31:Roll90Pitch180 32:Roll270Pitch180 33:Roll90Pitch270 34:Roll180Pitch270 35:Roll270Pitch270 36:Roll90Pitch180Yaw90 37:Roll90Yaw270	Overall board orientation relative to the standard orientation for the board type. This rotates the IMU and compass readings to allow the board to be oriented in your vehicle at any 90 or 45 degree angle. This option takes affect on next boot. After changing you will need to re-level your vehicle.
AHRS_RP_P	0,1		0.1 0.4	This controls how fast the accelerometers correct the attitude
AHRS_TRIM_X	-0,000976562	Radians	-0.1745 +0.1745	Compensates for the roll angle difference between the control board and the frame. Positive values make the vehicle roll right.
AHRS_TRIM_Y	0,02421117	Radians	-0.1745 +0.1745	Compensates for the pitch angle difference between the control board and the frame. Positive values make the vehicle pitch up/back.
AHRS_TRIM_Z	0	Radians	-0.1745 +0.1745	Not Used
AHRS_WIND_MAX	0	m/s	0 127	This sets the maximum allowable difference between ground speed and airspeed. This allows the plane to cope with a failing airspeed sensor. A value of zero means to use the airspeed as is.
AHRS_YAW_P	0,1		0.1 0.4	This controls the weight the compass or GPS has on the heading. A higher value means the heading will track the yaw source (GPS or compass) more rapidly.
ANGLE_MAX	4500	Centi-degre	1000 8000	Maximum lean angle in all flight modes
ARMING_CHECK	1		0:Disabled 1:Enabled -3:Skip Baro -5:Skip Compass -9:Skip GPS -17:Skip INS -33:Skip Params/Rangefinder -65:Skip RC 127:Skip Voltage	Allows enabling or disabling of pre-arming checks of receiver, accelerometer, barometer, compass and GPS
ATC_ACCEL_RP_MAX	0			
ATC_ACCEL_Y_MAX	0	Centi-Degre	0 720000:Disabled 18000:Slow 36000:Medium 54000:Fast	Maximum acceleration in yaw axis
ATC_RATE_FF_ENAB	0		0:Disabled 1:Enabled	Controls whether body-frame rate feedfoward is enabled or disabled
ATC_RATE_RP_MAX	18000			
ATC_RATE_Y_MAX	9000			
ATC_SLEW_YAW	1000	Centi-Degre	500 18000	Maximum rate the yaw target can be updated in Loiter, RTL, Auto flight modes
BAROGLTCH_ACCEL	1500			
BAROGLTCH_DIST	500			
BAROGLTCH_ENABLE	1			

Fig. 19: Mission Planner Full Parameter List

5.3 MISSION PLANNER CALIBRATION

Before an autonomous mission was run, a series of general calibrations were required for the flight controller to perform efficiently. The following three calibrations were performed, RC handheld controller calibration, accelerometer calibration, and compass calibration all of which are found under the SETUP tab in Mission Planner. The RC hand held calibration was necessary to checkout the different sticks on the controller to ensure the total range of throttle was utilized as well as setting the direction associated with change of throttle and identifying which stick was associated with position in the x and y direction and which was associated with the positive and negative z direction. Switches were also assigned for different flight modes and most importantly a switch was set to command a RTL. The compass calibration was very straight forward and involves enabling the calibration routine in Mission planner and then rotating the vehicle about the x, y, and z axis such that every side of the Quadcopter faces the center of the earth for a few seconds. The final calibration was the accelerometer calibration which calibrated the accelerometer within the CUBE autopilot's IMU. After being enabled and starting from a level surface, the vehicle is placed in the following sequence of positions as instructed by Mission Planner: Level, Left Side, Right Side, Nose Up, Nose Down, and Back Side. Figure 20 illustrates the positions associated with the accelerometer calibration procedure. After each position is set, the user pressed a confirmation button in Mission Planner before moving to the next position.

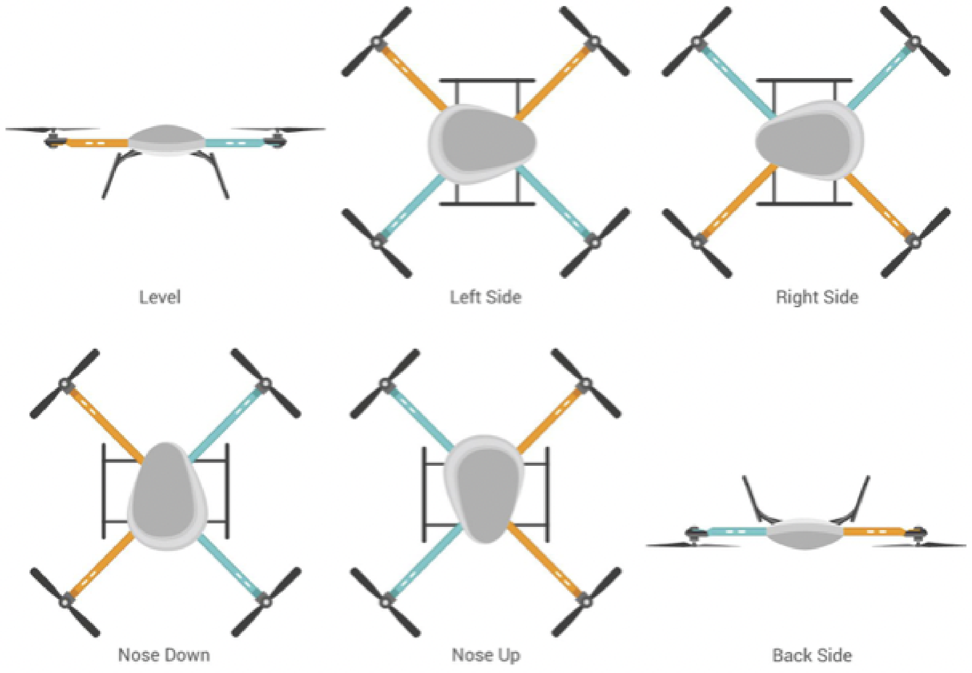


Fig. 20: Mission Planner Accelerometer Calibration Routine [6]

5.4 EXTERNAL IMU CALIBRATION AND DATA COLLECTION

A set of calibration coefficients were provided at the beginning of this research from a team that followed the simple calibration process mentioned in Section 2.2.2 in accordance with the process laid out in [20]. This provided additional calibration to the Analog Devices IMU's triaxial accelerometer. The ADIS 16475 manufacturers designed the IMU to output the sensor data registers applying unique formulas to correct the outputs for things like bias, sensitivity, misalignment, drift, and temperature independence. Reading the data registers was not an easy feature as the manufacturers only supplied data acquisition for when the IMU was physically connected to a computer with a Windows based operating system. As the RPI companion computer running the autonomous mission was of ARM architecture an alternative solution was necessary. C++ code found in Appendix C was written to read the data registers off the ADIS 16475 and apply the analog to digital conversion before writing them to a micro SD card on a Teensy board. This led to a relatively limited average sample rate of 300 hz. Data collection began just before the drone was armed to begin the autonomous mission and was completed after the mission was finished, at which point the drone was disarmed with the motors switched off.

5.5 PIXHAWK FLIGHT CONTROLLER LOGS

Data logs containing all of the different outputs available from the flight controller are stored on the autopilot and are made downloadable post flight. The data logs can be configured under the full parameter list found in the configuration tab of Mission Planner. For the purposes of this research, logs were written for each sequence of the drone being armed to being disarmed. The logs were critical in comparing the Analog devices IMU to the CUBE's IMU as all the various sensor outputs were recorded in the logs. The logs not only contained the IMU and GPS outputs but also the fused navigational readings produced from various optimization routines and on board EKFs. Figure 21 portrays a loaded flight log in mission planner with the IMU's acceleration data for the x, y, and z directions loaded.

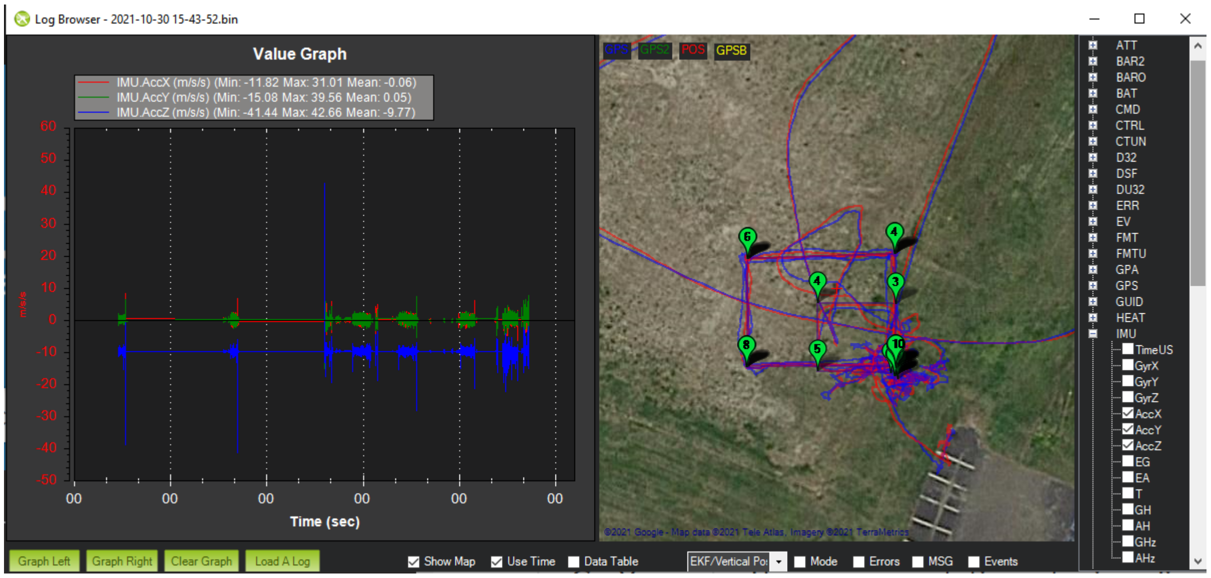
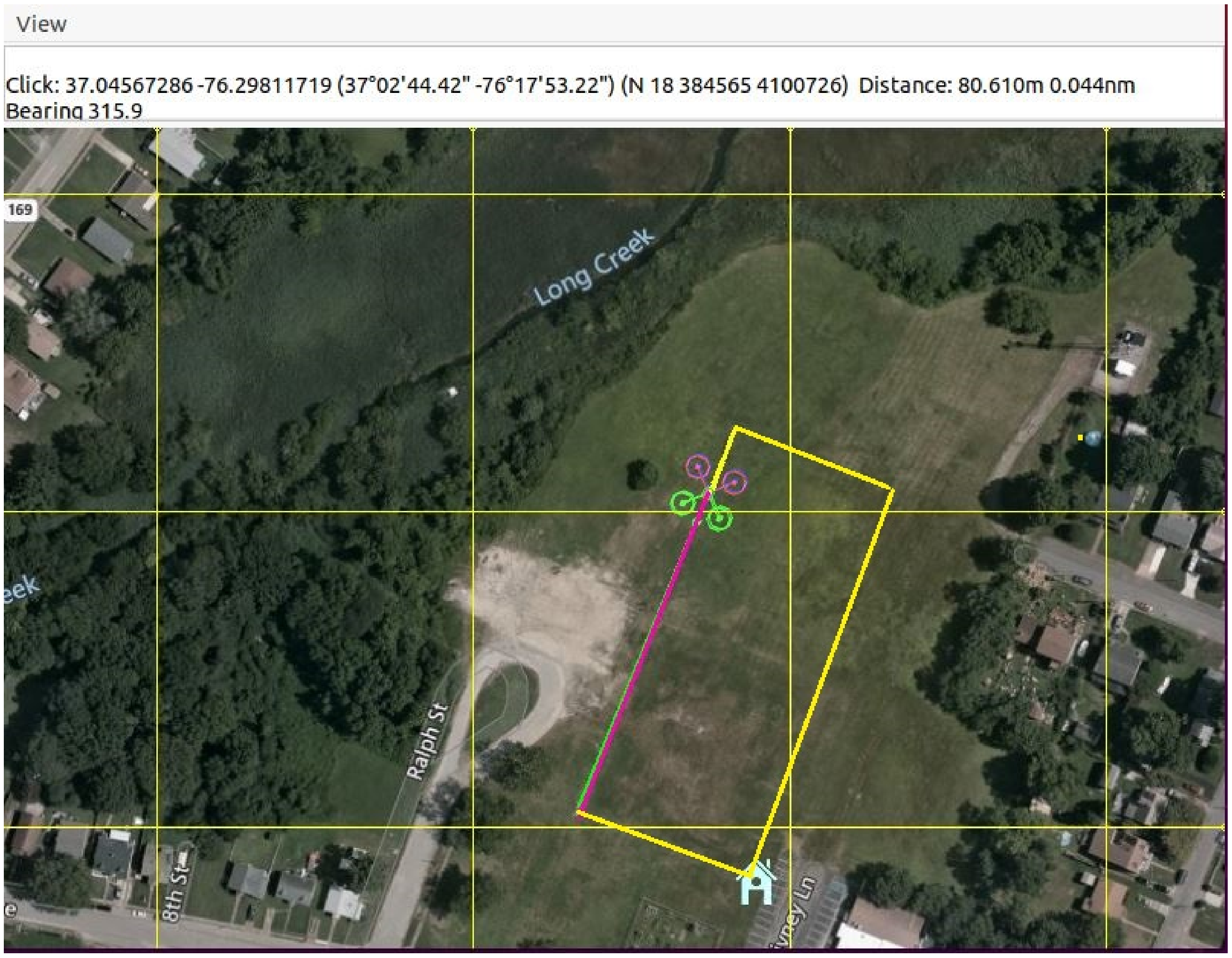


Fig. 21: Mission Planner Flight Log

5.6 FLIGHT MISSION

The flight mission was developed using ArduCopter in the simulation in the loop (SITL) environment as mentioned in section 4.1.2. A generic fight mission trajectory was defined loosely following that of a rectangle with the vertices represented by a GPS waypoint coordinates. A highly nonlinear trajectory was not used to test the hypothesis as it was outside the scope of this research. Each waypoint was approximately 15 meters apart and at an altitude of 15 meters. The commanded velocity for the mission was 5 m/s and the drone performed a position hold command at each waypoint for ten seconds before moving onto the next GPS coordinate. The trajectory was flown several times in support of various flight tests. In addition to the Python-based autonomous mission (Fly_mission.py), the same trajectory was run via the autonomous mission interface built into Mission Planner. This allowed the trajectory to be run on more than one drone in which a companion computer was not required. Figure 22 portrays the generic trajectory commanded in the autonomous missions developed in SITL.

Before initiating a flight mission, it was important to collect some ground data from


View

Click: 37.04567286 -76.29811719 (37°02'44.42" -76°17'53.22") (N 18 384565 4100726) Distance: 80.610m 0.044nm
Bearing 315.9

Fig. 22: Generic Flight Trajectory Developed in SILT Simulations (not to scale)

the external IMU to capture its performance with and without the vibrations from the Quadcopter's motors running. A comparison between the sets of data characterized the sensor measurement offsets. Different types of foam were also used to help dampen out the vibrations imposed by the motors through the frame and its interface with the external IMU. After several iterations of ground test runs were completed, the flight mission could be run with more confidence in the postflight analysis.

CHAPTER 6

RESULTS

6.1 1D RECTILINEAR MOTION

Figure 23 illustrates the artificial 1D acceleration data produced in Matlab that was corrupted with Gaussian white noise, used to verify the EKF algorithm.

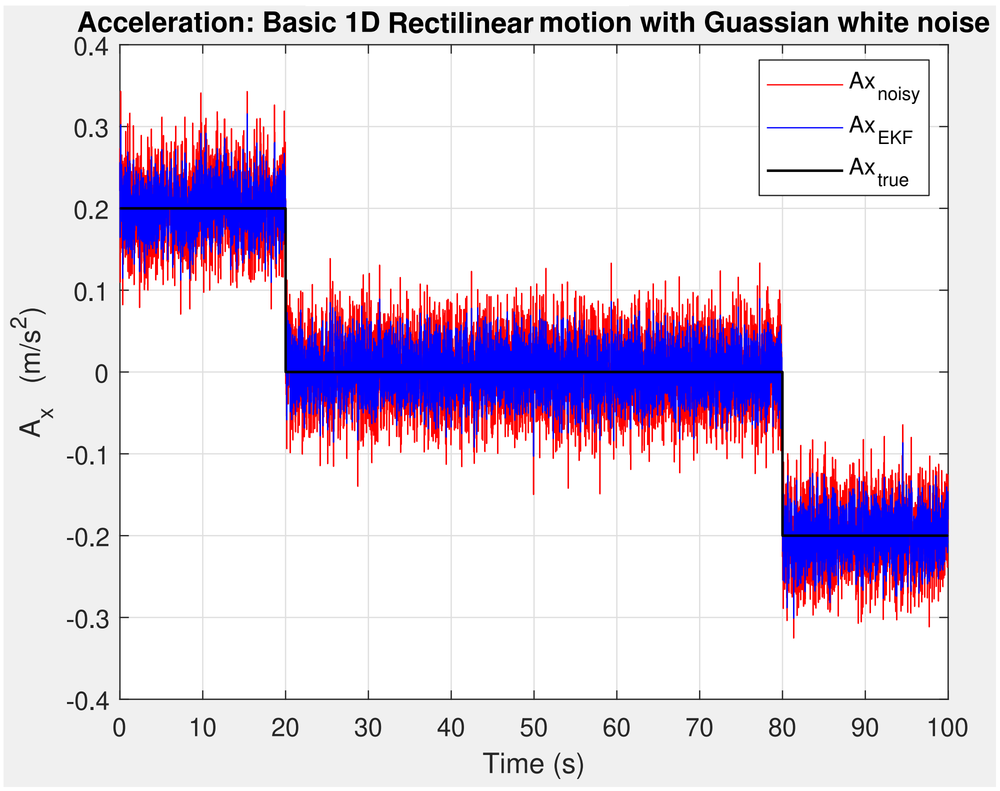


Fig. 23: 1D Rectilinear Motion Acceleration with a Signal to Noise Ratio of 20 (Gaussian White Noise)

Figure 24 portrays the integrated velocity of artificial 1D acceleration data produced (corrupted with Gaussian white noise) in Matlab and the EKF estimated velocity used for algorithm performance verification.

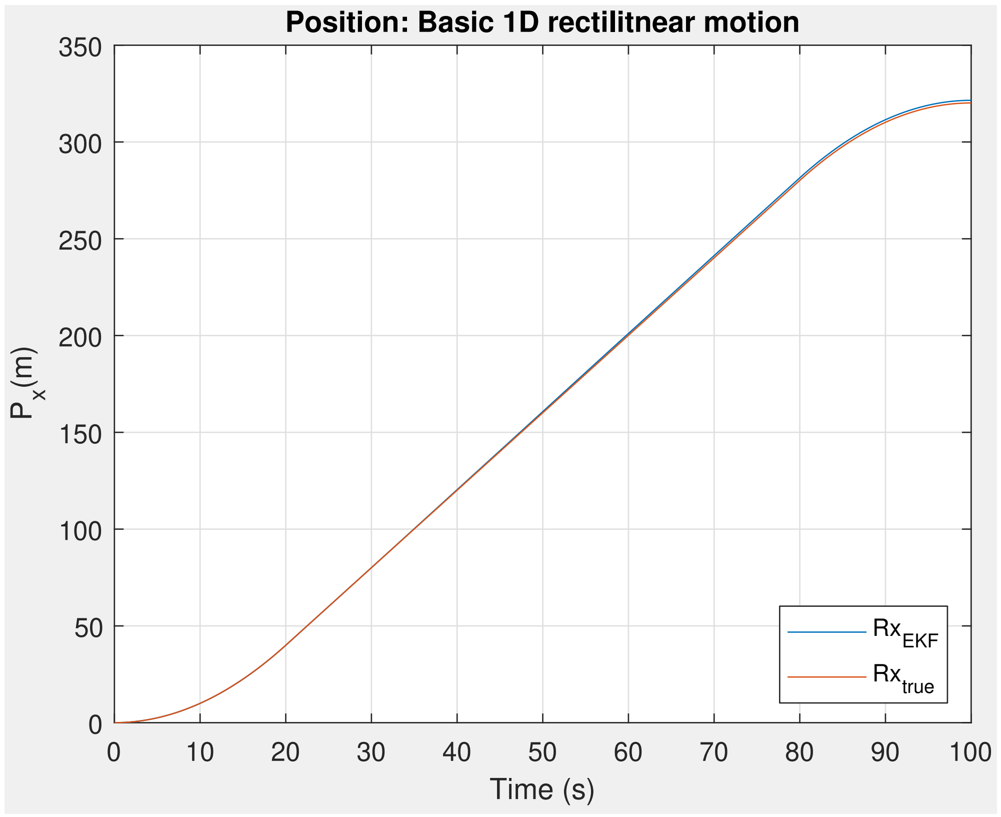


Fig. 24: 1D Rectilinear Motion Integrated Velocity

The figure above illustrates the integrated position of artificial 1D acceleration data produced (corrupted with Gaussian white noise) in Matlab and the EKF estimated velocity used for algorithm performance verification.

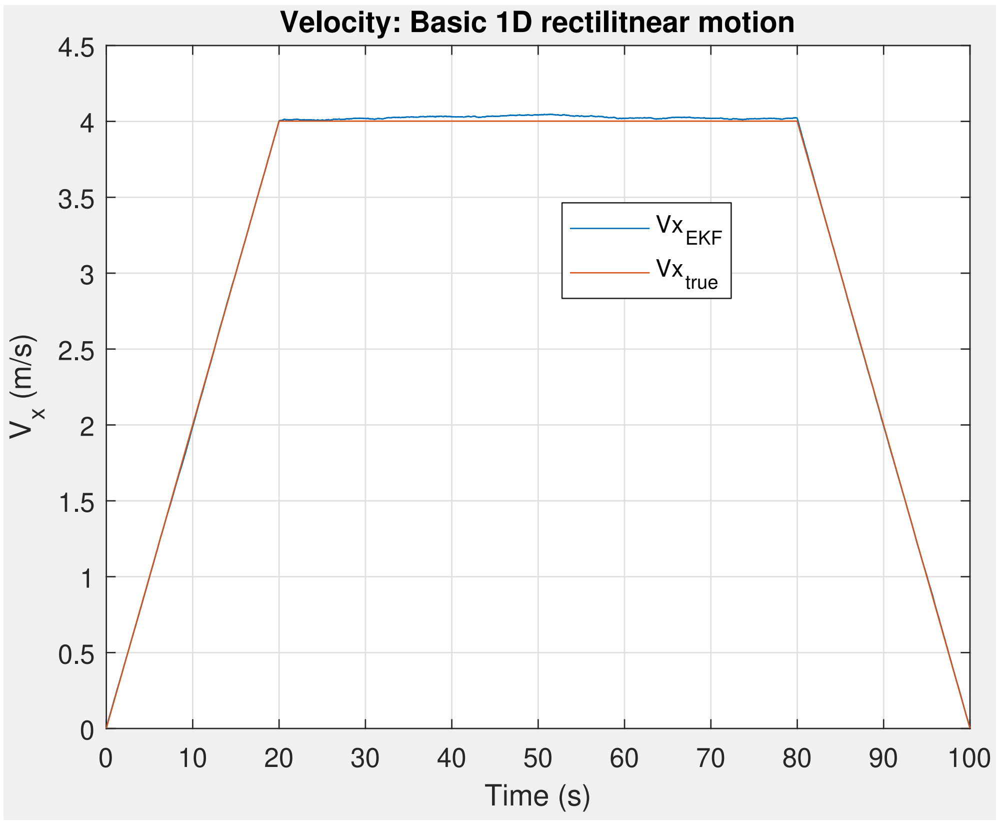


Fig. 25: 1D Rectilinear Motion Integrated Position

6.2 AUTOPILOT NAVIGATION DATA

Figure 26 is the autopilot's filtered x position converted from the Mission Planner GPS data (longitude).

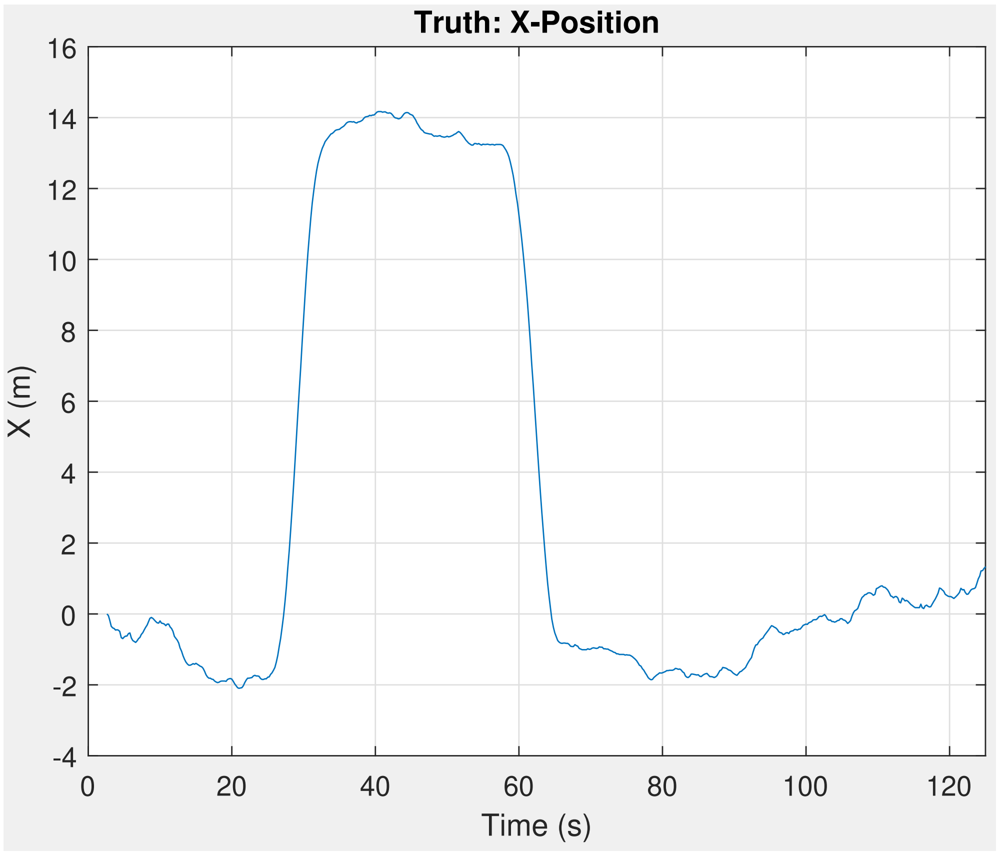


Fig. 26: Autopilot X-Position

Figure 27 is the autopilot's filtered y position converted from the Mission Planner GPS data (latitude).

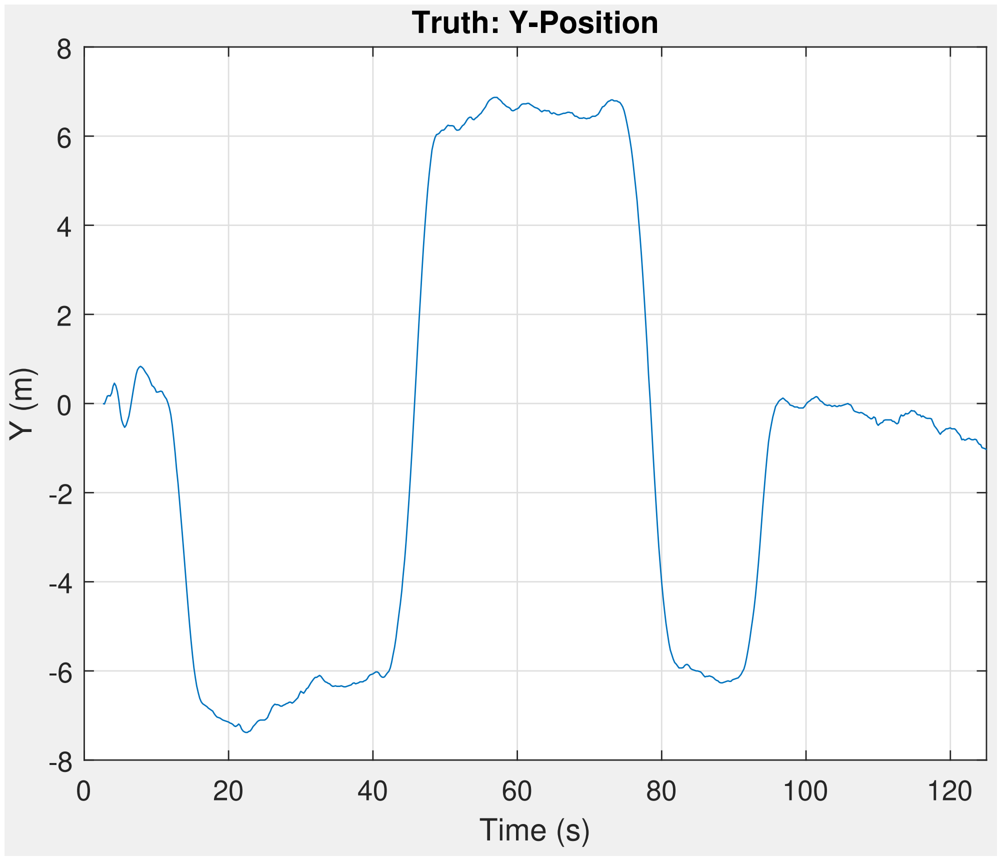


Fig. 27: Autopilot Y-Position

Figure 28 is the autopilot's filtered z position converted from the Mission Planner GPS data (altitude).

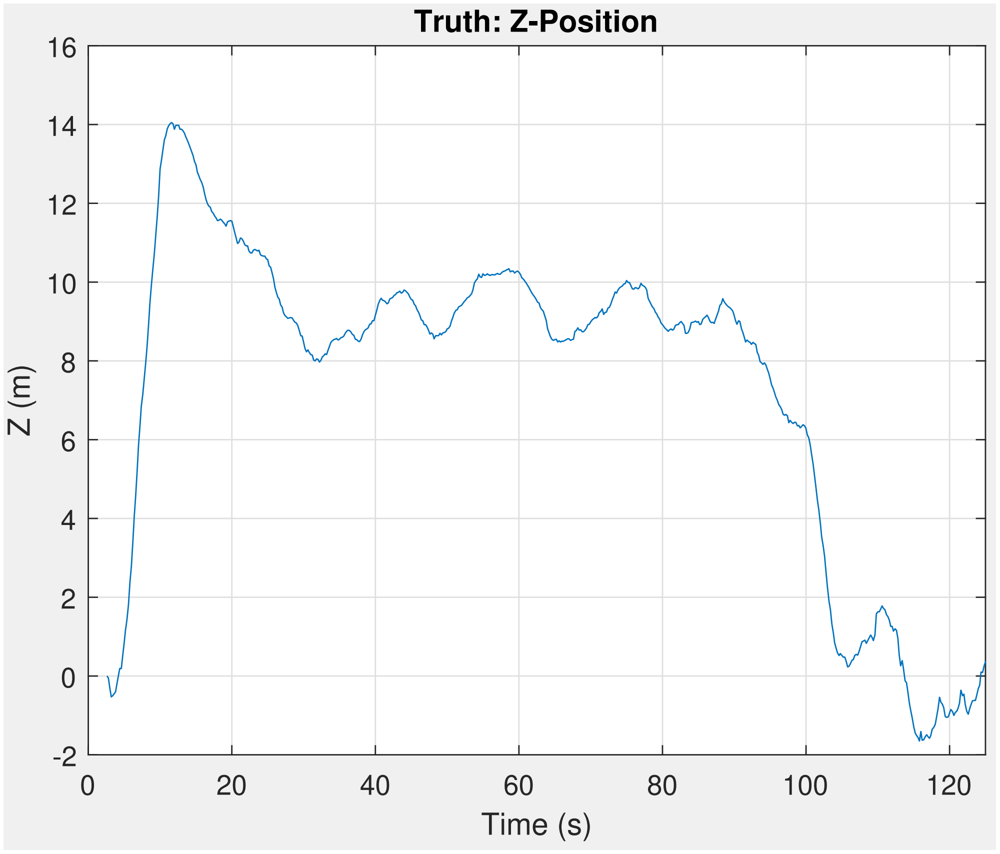


Fig. 28: Autopilot Z-Position

Figure 29 is the autopilot's filtered trajectory (x vs. y) converted from the Mission Planner GPS data.

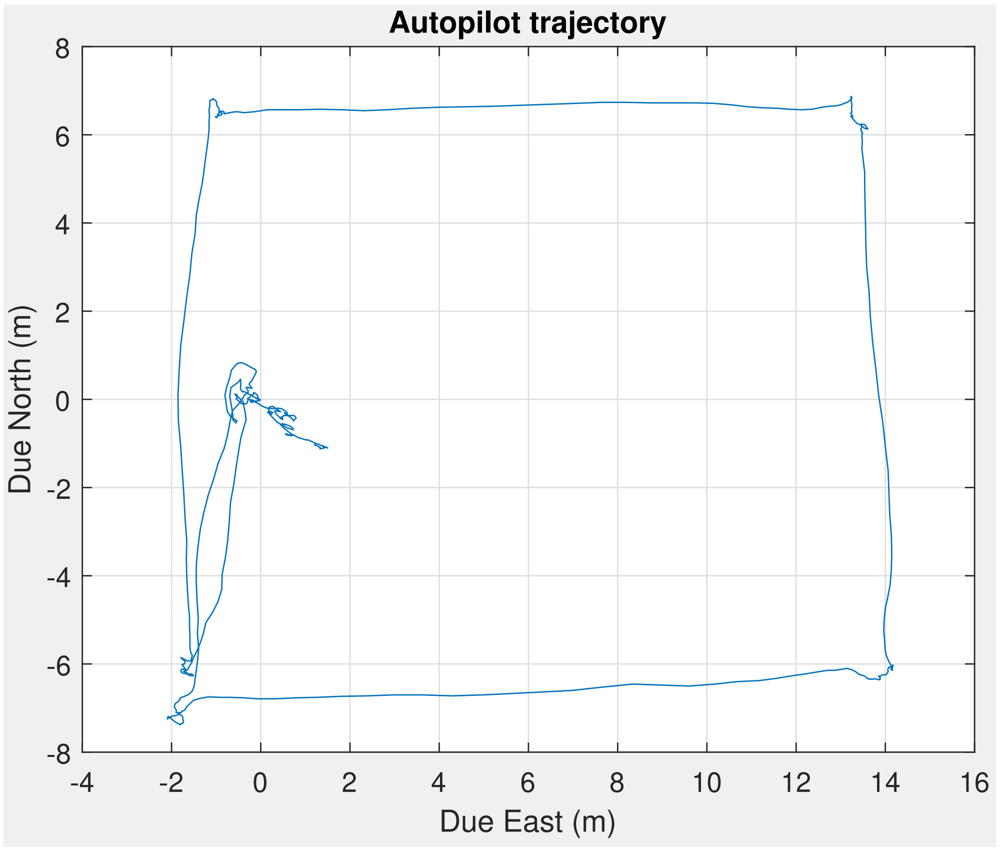


Fig. 29: Flight Trajectory

6.3 ANALOG DEVICES IMU

Figure 30 is the sensed angular rates about the x, y, and z axis from the Analog devices ADIS 16475 overlayed with the untuned EKF estimated values.

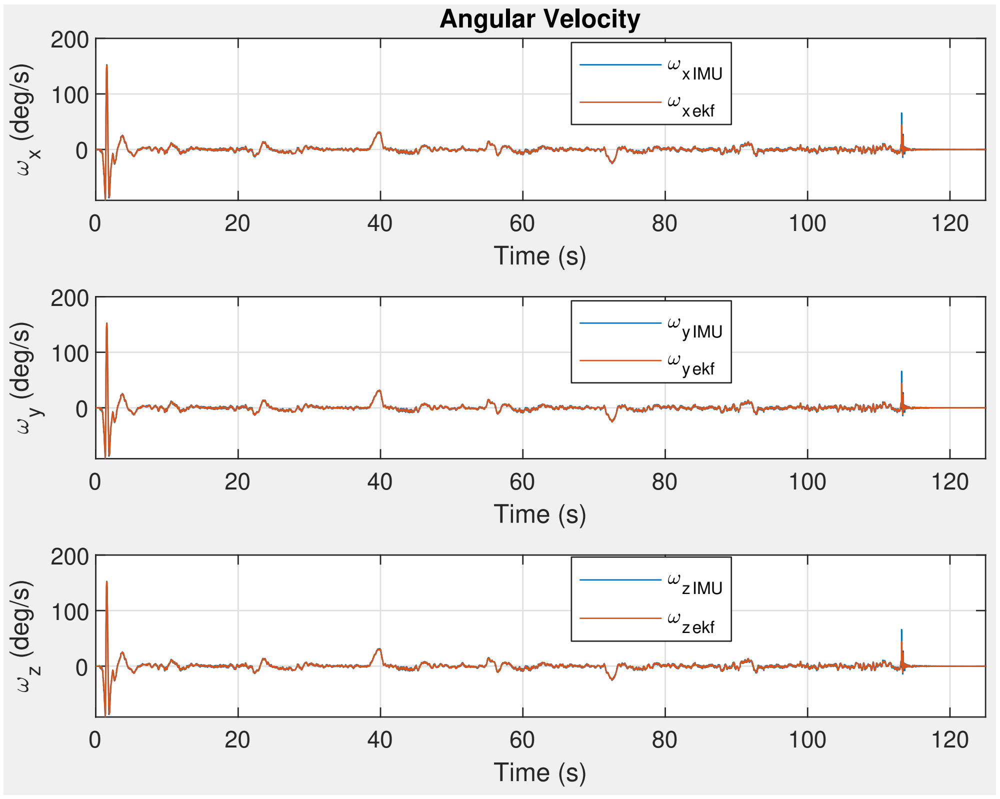


Fig. 30: Analog Devices IMU Angular Velocities

Figure 31 is the sensed orientation about the x, y, and z axis from the Analog devices ADIS 16475 overlayed with the untuned EKF estimated values.

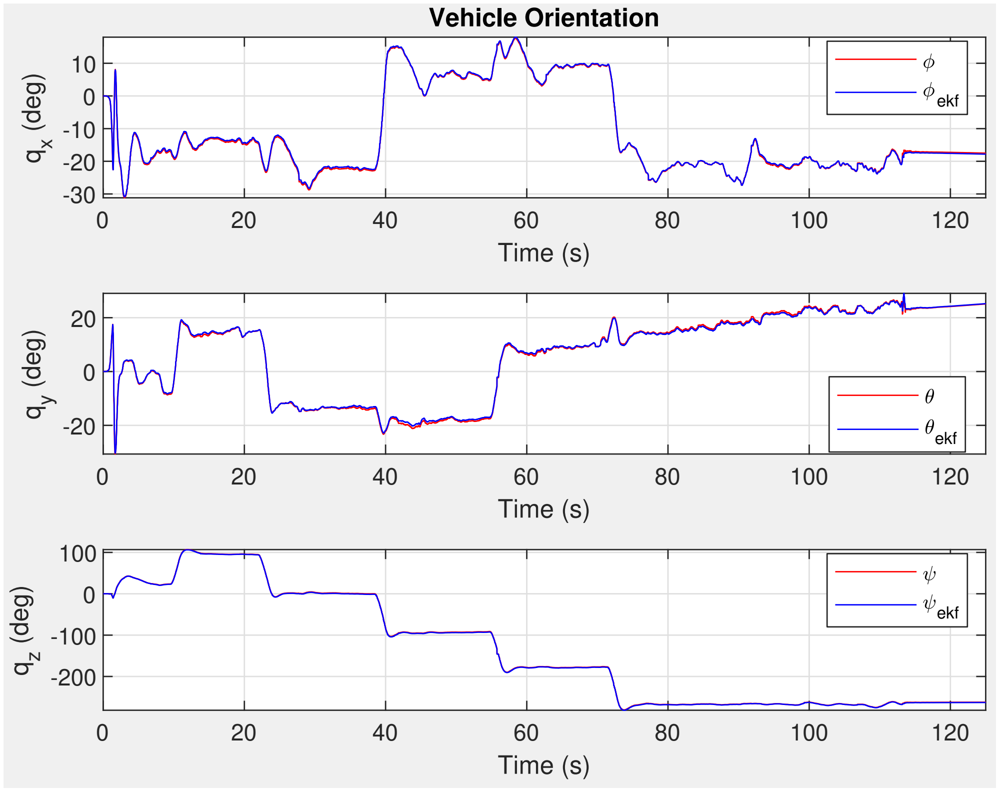


Fig. 31: Analog Devices IMU Angular Positions

Figure 32 is the sensed acceleration in the x direction from the Analog devices ADIS 16475 overlayed with the untuned EKF estimated values.

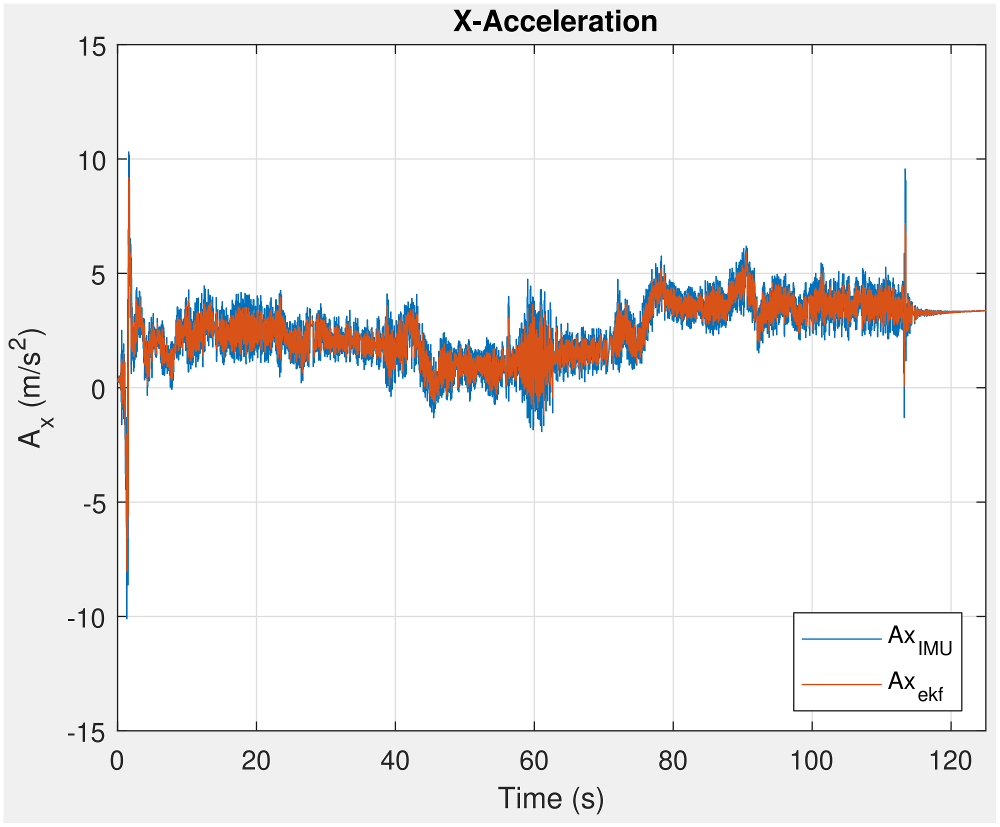


Fig. 32: Analog Devices IMU Acceleration in X Direction

Figure 33 is the sensed acceleration in the y direction from the Analog devices ADIS 16475 overlayed with the untuned EKF estimated values.

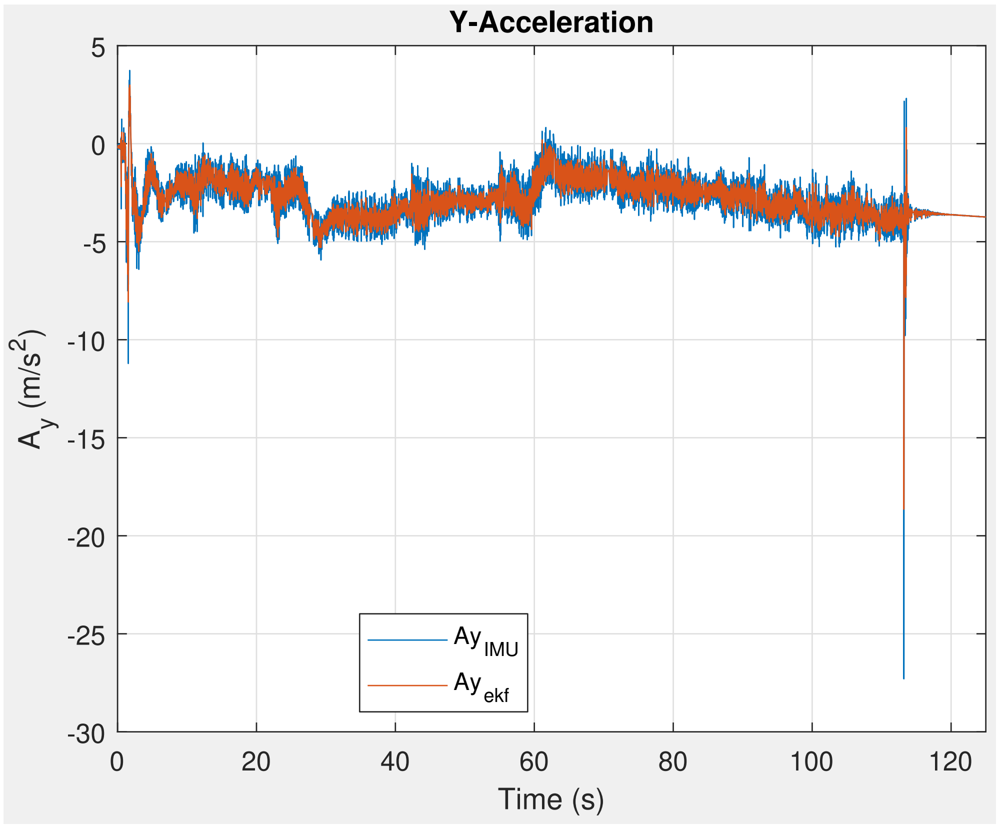


Fig. 33: Analog Devices IMU Acceleration in Y Direction

Figure 34 is the sensed acceleration in the z direction from the Analog devices ADIS 16475 overlayed with the untuned EKF estimated values.

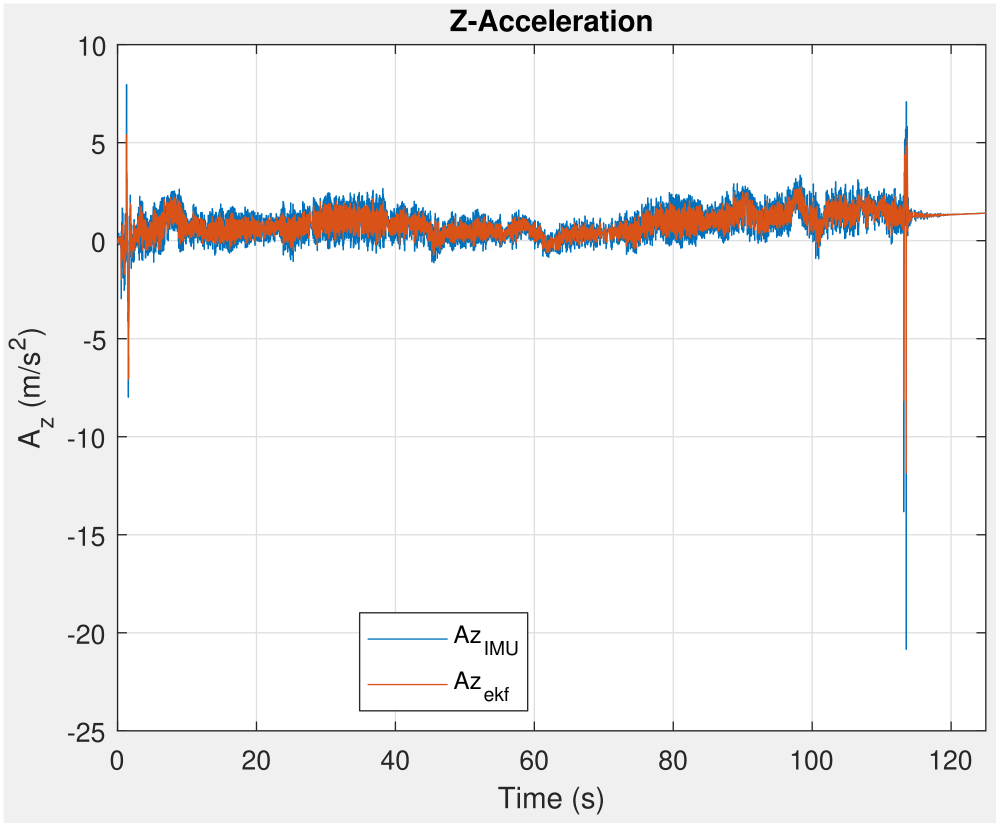


Fig. 34: Analog Devices IMU Acceleration in Z Direction

Figure 35 is the integrated velocities in the x, y, and z directions from the sensed Analog devices ADIS 16475 accelerations overlayed with the untuned EKF estimated values.

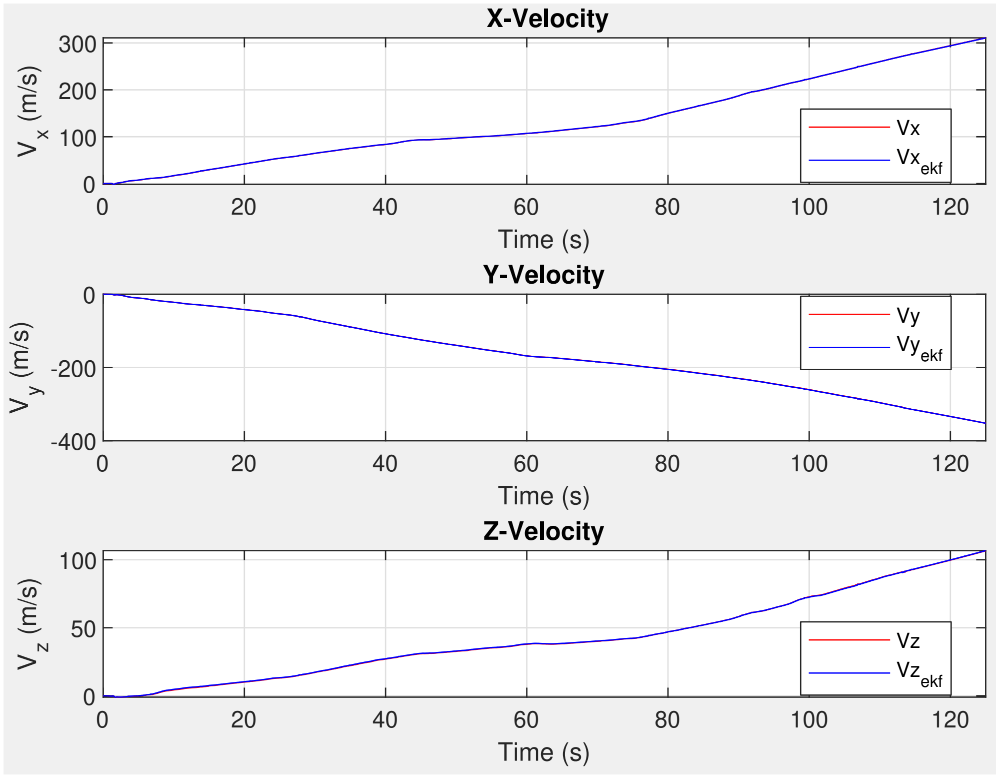


Fig. 35: Analog Devices IMU Integrated Velocities

Figure 36 is the integrated position in the x, y, and z directions from the sensed Analog devices ADIS 16475 accelerations overlayed with the untuned EKF estimated values.

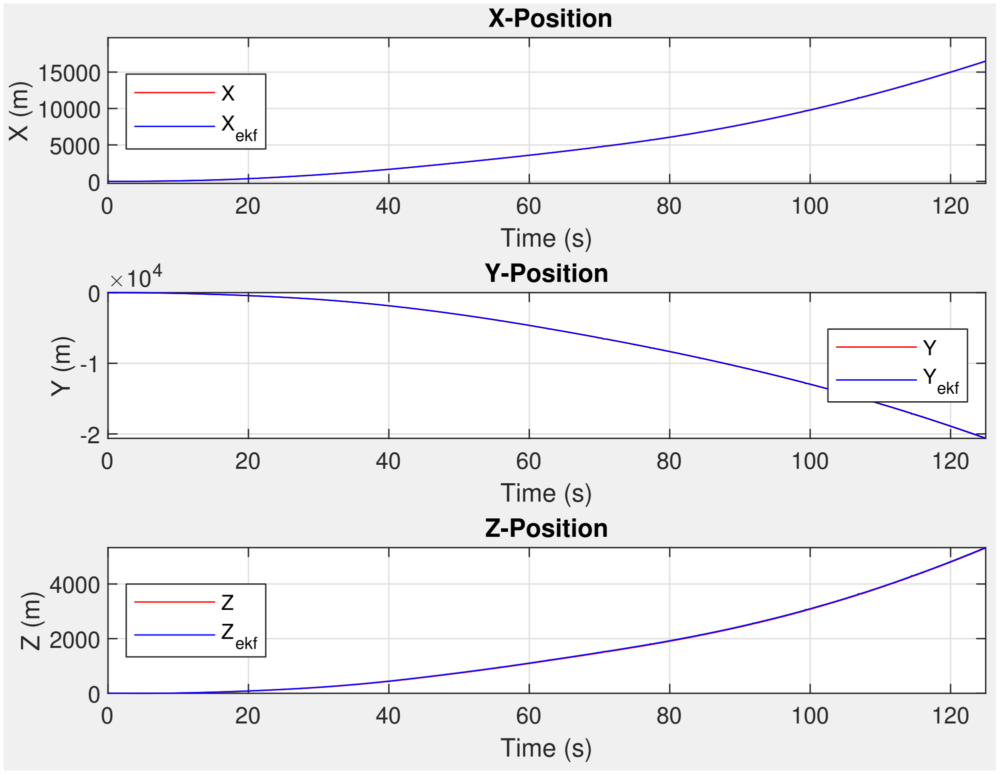


Fig. 36: Analog Devices IMU Integrated Positions

6.4 EKF INERTIAL NAVIGATION ESTIMATES

Figure 37 is the sensed angular rates about the x, y, and z axis from the Analog devices ADIS 16475 overlayed with the tuned EKF estimated values.

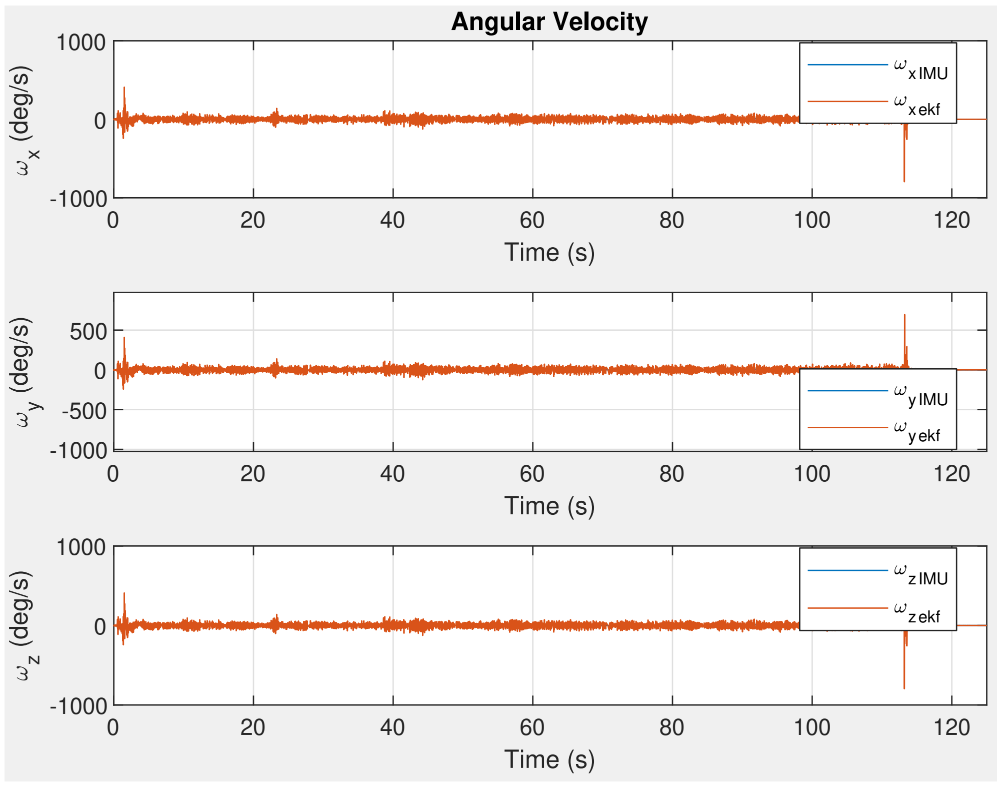


Fig. 37: Analog Devices Tuned IMU Angular Velocities

Figure 38 is the sensed orientation about the x, y, and z axis from the Analog devices ADIS 16475 overlayed with the tuned EKF estimated values.

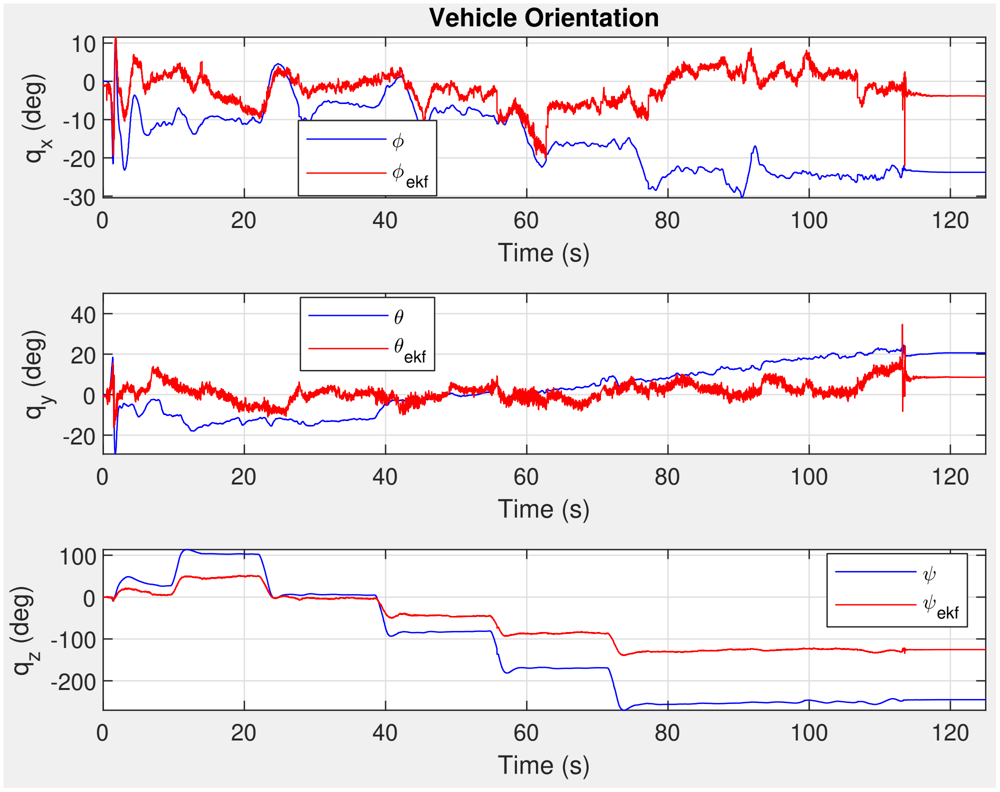


Fig. 38: Analog Devices Tuned IMU Angular Positions

Figure 39 is the sensed acceleration in the x direction from the Analog devices ADIS 16475 overlayed with the tuned EKF estimated values.

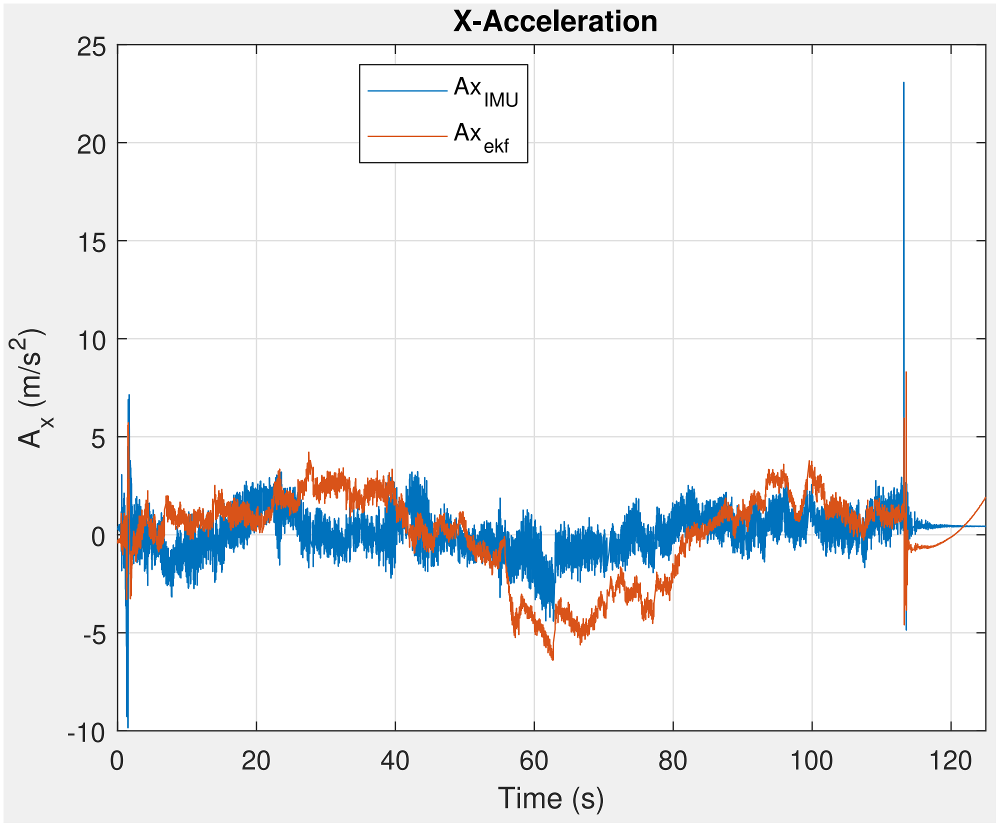


Fig. 39: Analog Devices Tuned IMU Acceleration in X Direction

Figure 40 is the sensed acceleration in the y direction from the Analog devices ADIS 16475 overlayed with the tuned EKF estimated values.

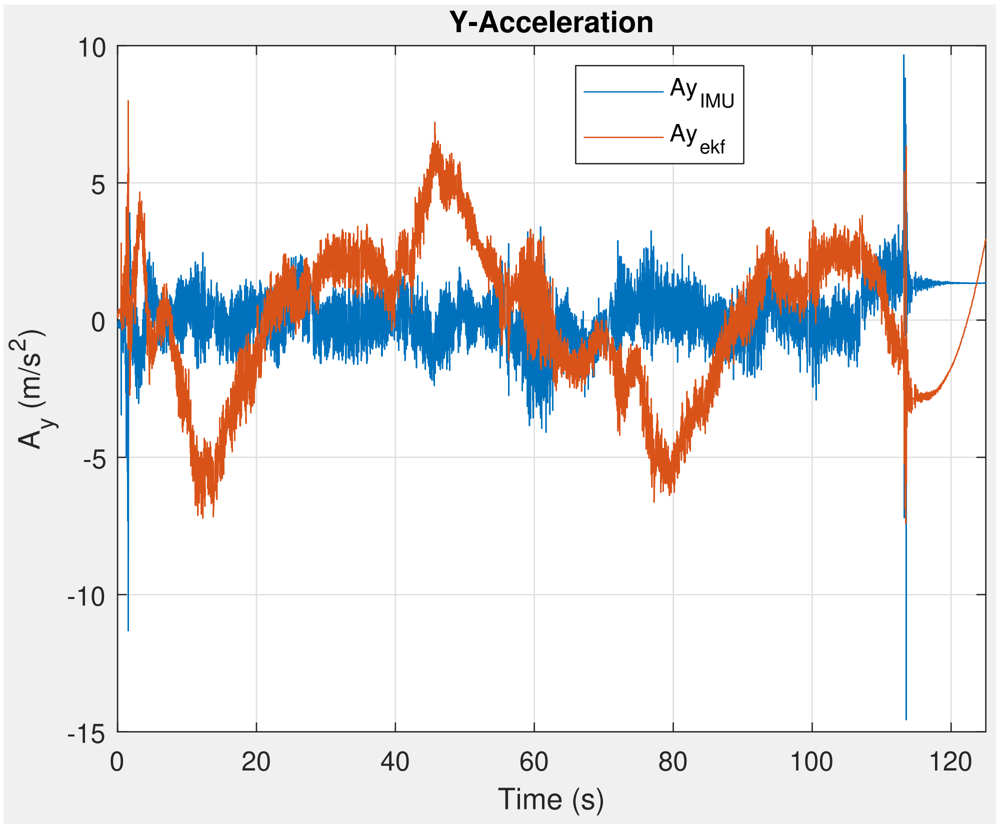


Fig. 40: Analog Devices Tuned IMU Acceleration in Y Direction

Figure 41 above is the sensed acceleration in the z direction from the Analog devices ADIS 16475 overlayed with the tuned EKF estimated values.

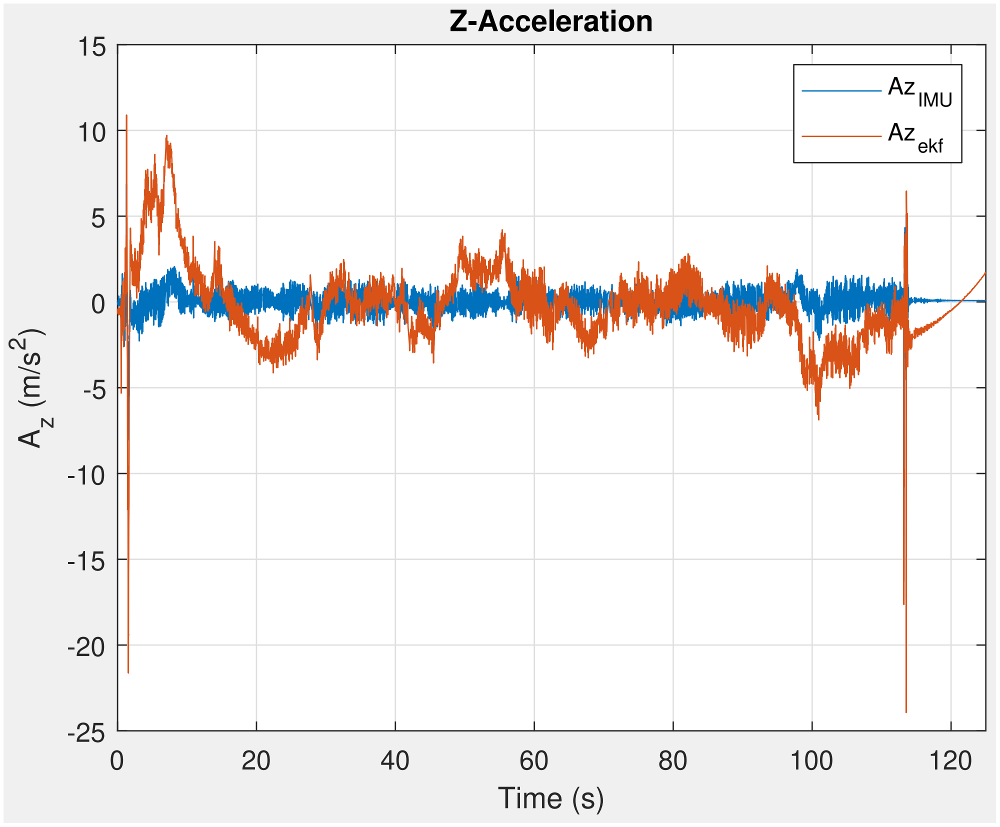


Fig. 41: Analog Devices Tuned IMU Acceleration in Z Direction

Figure 42 is the tuned velocity estimates in the x, y, and z directions from the EKF filtered Analog devices ADIS 16475.

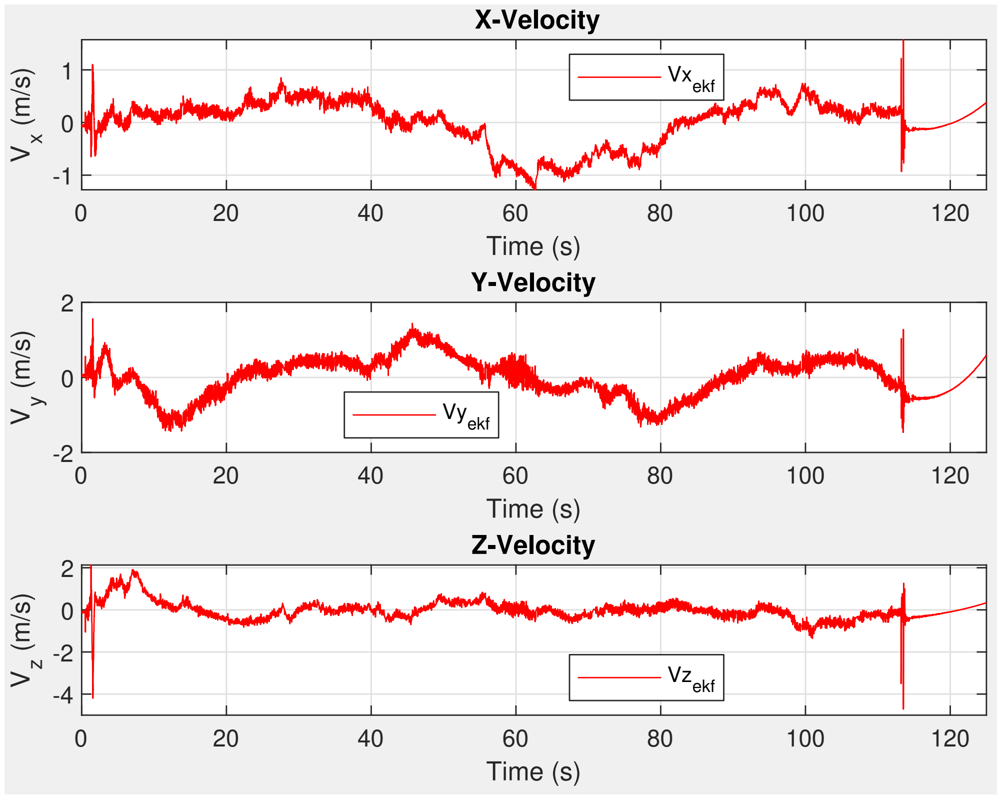


Fig. 42: Analog Devices Tuned IMU Integrated Velocities

Figure 43 is the tuned position estimates in the x, y, and z directions from the EKF filtered Analog devices ADIS 16475.

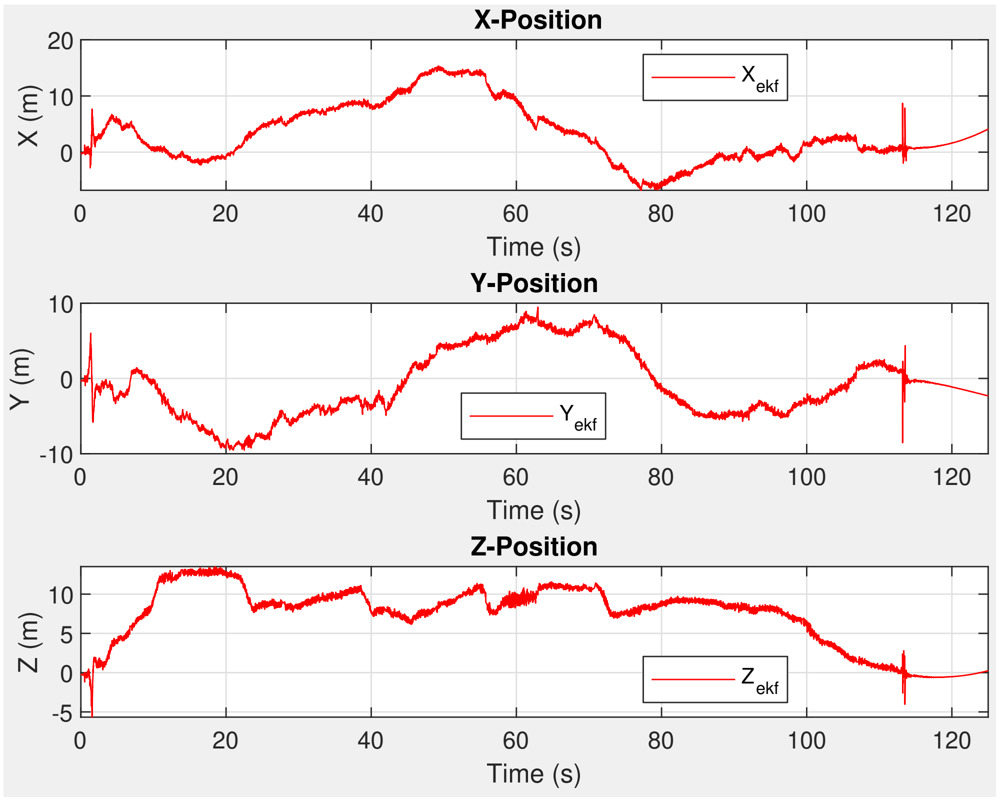


Fig. 43: Analog Devices Tuned IMU Integrated Positions

Figure 44 overlays the final tuned trajectory estimate with a partially tuned trajectory estimate illustrating the impact of the optimized covariance matrices.

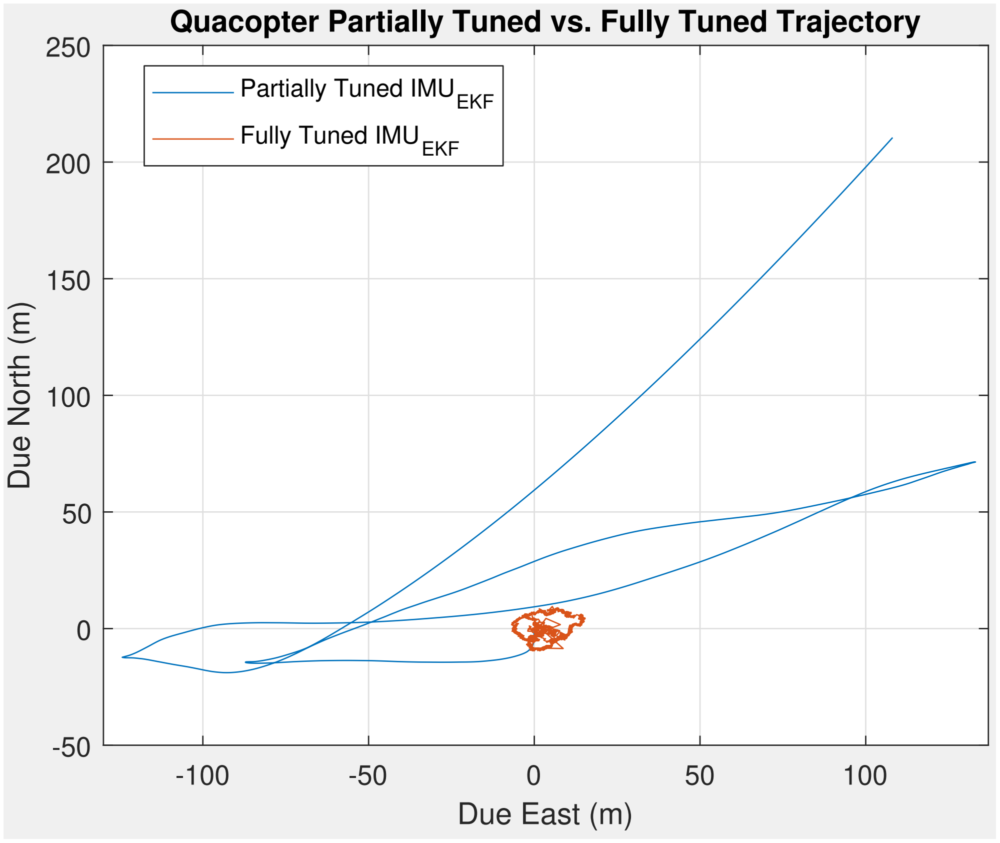


Fig. 44: Flight Trajectory Filtered vs Partial Filter Comparison

Figure 45 is the tuned flight trajectory estimate based on the measurements of the Analog devices ADIS 16475 IMU versus that of the autopilot.

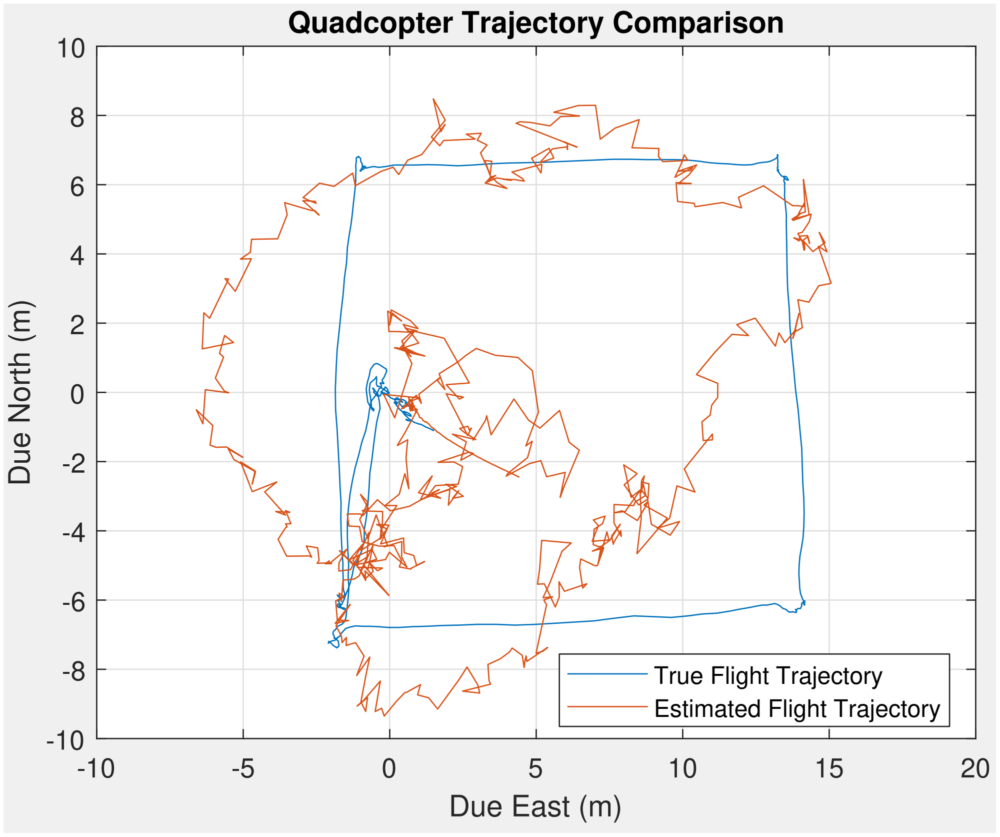
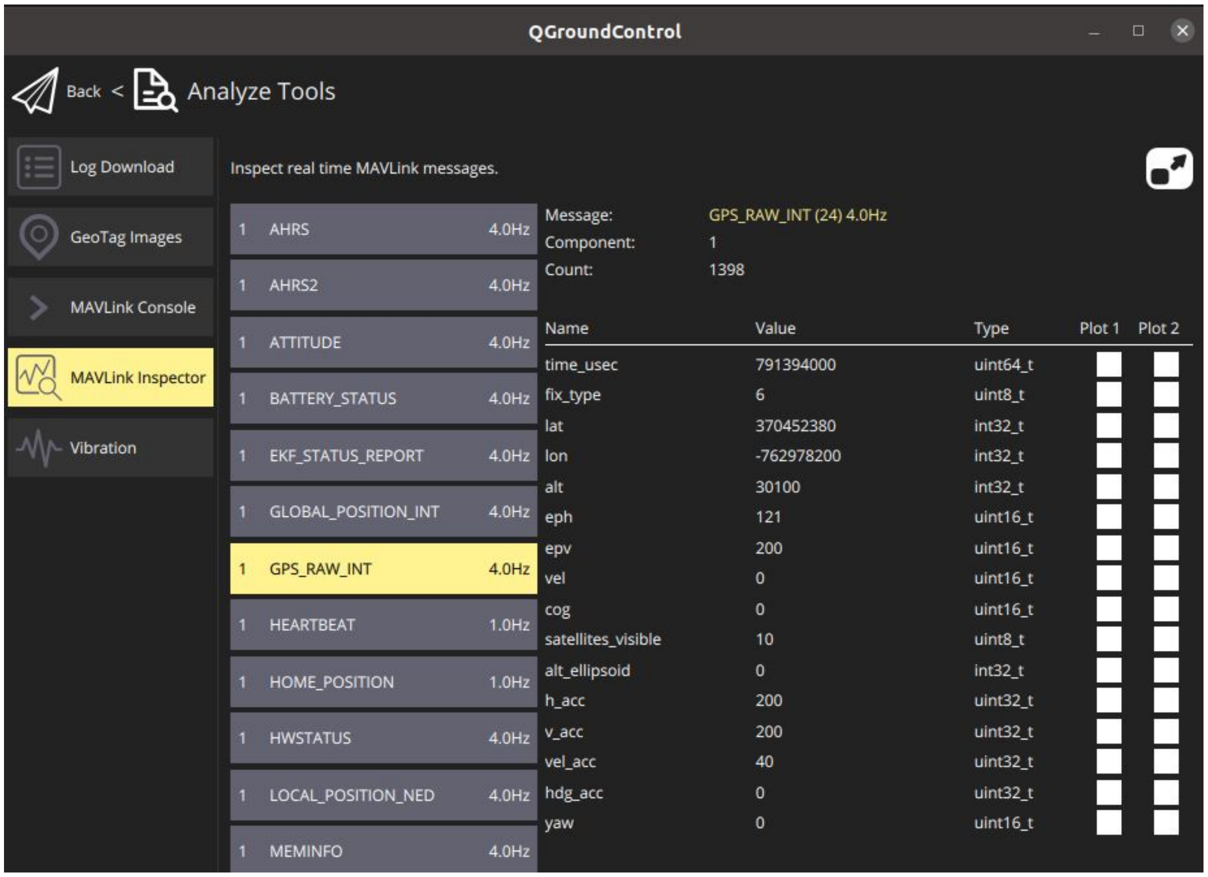


Fig. 45: Flight Trajectory Filtered vs Truth Comparison

6.5 QGROUNDCONTROL GPS SPOOFING

Figure 46 is QGroundControl running in SITL demonstrating the spoofed GPS values being read by the Quadcopter's Autopilot



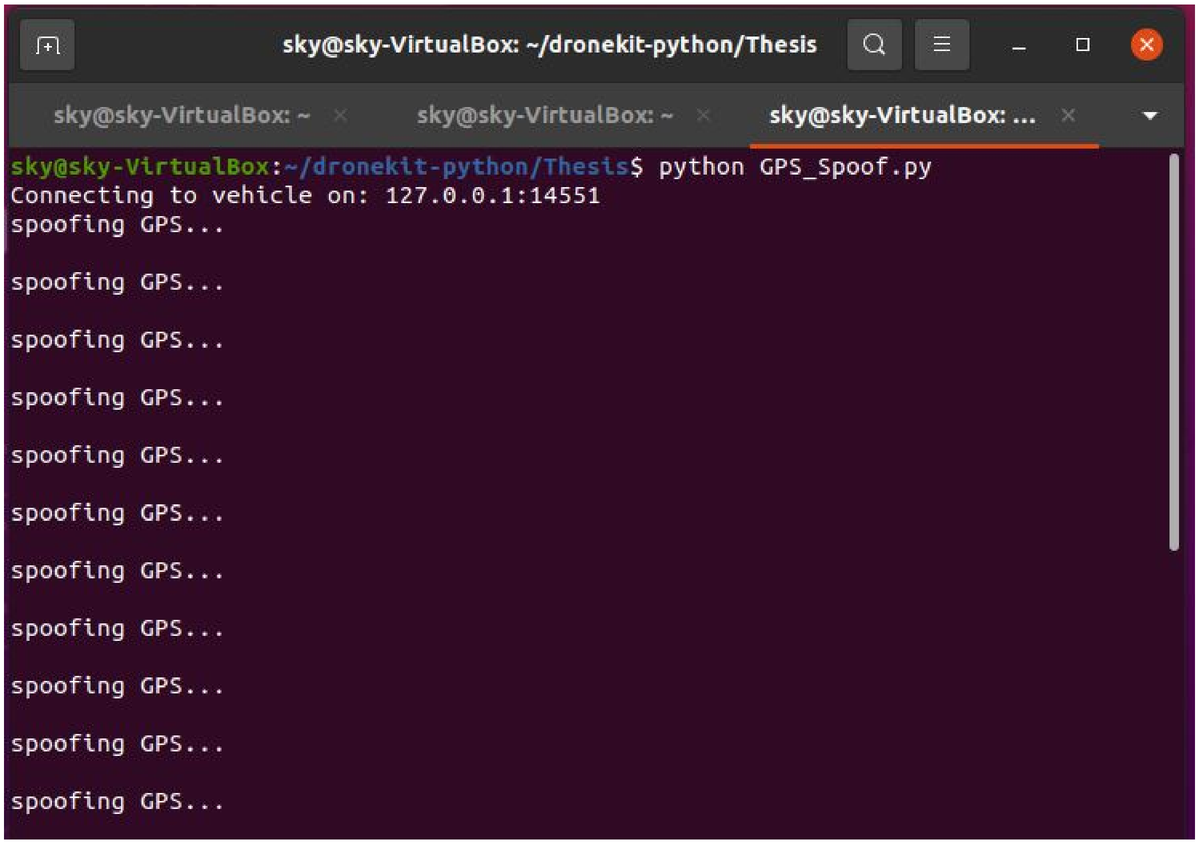
Inspect real time MAVLink messages.

Message:	GPS_RAW_INT (24) 4.0Hz
Component:	1
Count:	1398

Name	Value	Type	Plot 1	Plot 2
time_usec	791394000	uint64_t	☐	☐
fix_type	6	uint8_t	☐	☐
lat	370452380	int32_t	☐	☐
lon	-762978200	int32_t	☐	☐
alt	30100	int32_t	☐	☐
eph	121	uint16_t	☐	☐
epv	200	uint16_t	☐	☐
vel	0	uint16_t	☐	☐
cog	0	uint16_t	☐	☐
satellites_visible	10	uint8_t	☐	☐
alt_ellipsoid	0	int32_t	☐	☐
h_acc	200	uint32_t	☐	☐
v_acc	200	uint32_t	☐	☐
vel_acc	40	uint32_t	☐	☐
hdg_acc	0	uint32_t	☐	☐
yaw	0	uint16_t	☐	☐

Fig. 46: QGroundControl Spoofed GPS Data

Figure 47 illustrates a command window print out associated with the execution of the GPS_Spoof.py script in the companion computer's environment



```
sky@sky-VirtualBox:~/dronekit-python/Thesis$ python GPS_Spoof.py
Connecting to vehicle on: 127.0.0.1:14551
spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...

spoofing GPS...
```

Fig. 47: GPS_Spoof.py Running in SITL Environment

CHAPTER 7

DISCUSSION

Throughout the course of this research, the intention was to apply an Extended Kalman Filter introduced in Section 2.3 and test the effectiveness of the MMM method on inertial measurements collected with a higher quality, ADIS 16475, IMU. The Quadcopter built for flight test and development runs with a Cube black Ardupilot based autopilot that relies on its suite of sensors to perform autonomous missions. In order to ensure autonomous capability the GPS measurement must be available. The measurements from the external IMU were passed through an EKF to predict the inertial position of the Quadcopter. The predicted position values can then be passed to the autopilot as GPS coordinates to ensure autonomous mission capability can be maintained in situations when the physical GPS measurements may not be authentic or available. The findings discussed within this section are broken into four parts, EKF MMM algorithm verification, a review of the initial flight test data, improvements to the navigation estimates, and spoofing the GPS via MAVLink messaging.

7.1 EKF INERTIAL NAVIGATION VERIFICATION

A Kalman Filter is a state estimator that operates on the basis that a random function of time may be thought of as an output of a dynamic system excited by an independent Gaussian random process. The Extended Kalman Filter is ideal for a non-linear process or non-linear measurement relationship. The EKF for the state-space system associated with the MMM algorithm for this research needed to be verified before attempting to estimate state information from the ADIS IMU measurements. To ensure the EKF worked artificial one dimensional accelerometer data was produced with Matlab. As this was an idealized scenario, the fake measurements were corrupted with Gaussian white noise, satisfying the underlining assumption of the state estimator. The corrupted one dimensional accelerometer data had a signal to noise ration of 10 decibels.

The data used to verify the EKF routine were characteristic of a one dimensional rectilinear motion. The artificial accelerations were intended to reflect the motion of a point mass speeding up from rest, then translating at a constant velocity, and finally slowing down to a velocity of 0 ft/s. Fundamental first principals were used to calculate the position and velocity as illustrated in Equations 44 and 45.

$$d = v_i t + \frac{1}{2} a t^2 \tag{44}$$

$$v_f = v_i + at \tag{45}$$

Figure 23, presented in the previous section shows the unfiltered acceleration corrupted with white noise overlayed with the filtered acceleration. Clearly the Extended Kalman Filter is estimating a less noisy signal. To verify the state estimator the results produced from the Matlab routine were compared against the theoretical position and velocity described in Equations 44 and 45. The navigation data presented in Figures 24 and 25 illustrate the predicted position and velocity from the EKF routine matching those calculated using the equations mentioned previously. These results gave confidence in the algorithm's accuracy and performance.

7.2 INITIAL FLIGHT TEST

As mentioned in Section 5.6, several flight test were required for this research. As one could expect the first round flight test produced poor data. The ADIS 16475, while extremely more sensitive than the INS provided in the CUBE autopilot, was still highly susceptible to noise and bias. The factory calibration provided as well as initial bias correction maintained from previous works was not sufficient enough to eliminate the fundamental drift attributed to DC offset or due to the vibrations imposed on the rigid body of the vehicle. Figures 32 through 34 illustrate the sensed accelerations overlayed with the filtered estimates produced with by the EKF. As conveyed in Figures 35 and 36 velocity and position measurement were unreliable in reflecting the true position of the vehicle due to the error and bias creeping into the integration process. Various attempts were made to adjust for such interference and implemented with additional flight test and analysis.

The first attempts at reducing noise from the ADIS 16475 measurements were to reduce the impact of vibration and to account for the bias or offset in measurements. Different locations of the sensor were considered as well as different mounting techniques. Ultimately a 3M foam was used to interface the IMU to the frame as suggested by the autopilot manufacturers at the center of gravity (CG) location approximately equidistant from the rotors. Assumptions were made regarding the bias in that it was a constant without any drift. Further flight tests led to the conclusion that the bias did suffer from a form of drift. It is not uncommon for strapdown inertial navigation systems to struggle from such errors. In fact very few system are actually capable of provided accurate and reliable inertial navigation estimates.

The main errors associated with the IMU stemmed from scale factor, bias, misalignment, and nonlinearities in the sensed values. A reliable approach in accounting for these errors is an all encompassing calibration routine and modeling the errors in the state space model. For this research, the MMM EKF method does not include such states and therefore the validity of its performance objectively had to be tested. State estimation is also hindered by the inherent errors associated with numerical integration. While, in principle, numerical integration is not complicated, the difference between actual position and orientation increase overtime. The phenomenon is known as integration drift and it is unavoidable. Integration drift and the presence of noise in the IMU measurements exacerbate the effects and the deviation occurs more rapidly and on a larger scale. For this matter, fusing a GPS measurement with an IMU measurement as an a priori navigational estimate is suitable for eliminating the error. In such systems the GPS navigational readings are subtracted from the integrated INS navigational readings whenever GPS is available.

7.3 EKF NAVIGATION ESTIMATION

After all of the improvements were made to the experimental set up to aid the quality in measurement of the external IMU navigational estimates were still unacceptable. The final attempts at salvaging near successful results with the MMM EKF were made in tuning the process and measurement noise covariance matrices. For this tuning non-linear optimization routines know industry wide were applied. The two used for this research were the FMINCON and Simulated Annealing algorithms. Treating the navigation data from the autopilot as truth data, the covariance matrices were tuned in hopes of producing improved estimates for similar flight trajectories.

FMINCON and Simulated Annealing are non-linear optimizer algorithms that find the minimum of a given function. FMINCON was found to not be ideal for this research as it is prone to stopping after reaching a local minima. Rather, the simulated annealing algorithm was chosen. This algorithm is more computationally expensive and has several more iterations but, it is proficient at finding the global minima. The cost function used for the simulated annealing algorithm was based on the differences between the filtered navigational position produced with the MMM EKF and the navigation data of the on board autopilot (predictions from the suit of sensors, including GPS, and its own set of EKF filtering). The cost function to be minimized is given in Equation 46

$$\sum((x_{predicted} - x_{truth})^2 + (y_{predicted} - y_{truth})^2 + (z_{predicted} - z_{truth})^2) \tag{46}$$

The simulated annealing algorithm was executed four time via a Matlab script, each time for 30,000 iterations holding onto the best performing inputs which minimized the cost function every 100 iterations. The elements of the covariance matrices served as inputs to the MMM EKF which produced the tuned state estimates (predictions for Equation 46). An initial guess of $.001 * I(15)$ was used for measurement and process noise covariance matrices. The MMM EKF method was applied for each iteration. Then the state estimates were compared against the states recorded by the autopilot via a root sum square. The simulated annealing algorithm would next vary the inputs to the covariance matrices and repeat the iteration. Through the applications of the simulated annealing algorithm the following matrices were determined:

```
Q =

  Columns 1 through 8

   0.999816619693786   0.000619537003707   0.000210098620369  -0.000354284889322   0.000975760582669   0.000791286173773  -0.001525106374904  -0.001248885709449
   0.000271555355109   1.000495263448180  -0.000090955068547  -0.000018036336446  -0.000030612151963  -0.000156501735342   0.002327253774538  -0.000606632393085
  -0.000093870491049  -0.000343346660076   0.999046993248305   0.000301195664916   0.001010138342128  -0.000078947195283   0.002269066545593  -0.001674651334804
   0.000173964406919  -0.000615663614031   0.000056980059547   0.999071081955287  -0.000292824215963  -0.000058261555487   0.001885938553197  -0.000455535123112
   0.000667424763934  -0.000332814313924   0.000134391578938   0.000315657901015   1.000061700864814   0.000552830531476   0.000843248549491  -0.000228269787630
   0.000683768428158  -0.000173960199867  -0.000894143467796   0.000383558069023   0.000478148551797   1.000078151615931  -0.000538457465917   0.003696645845358
   0.000489375815635   0.000522232422240  -0.000098888540359  -0.000403406023453  -0.000312751542361  -0.000426035711177   1.001030494546166   0.003075921610446
  -0.000638824413158  -0.000039781215716   0.000170245832786   0.000626438735659   0.000150357913958   0.000338845723199  -0.002344218682392   1.001407065918822
   0.001136184325541   0.000018655241191  -0.000090348715215  -0.000193164502561  -0.000000286110444   0.000404891445863   0.003551111137886  -0.003365778355615
  -0.000425534582546   0.000261133687852   0.000376671292088  -0.000327788957229   0.000539279379146   0.000776544670539   0.001953500390858   0.014914412568584
   0.000924208087324   0.000256187352852  -0.000076269191184   0.000928187188640   0.000563087719598   0.000664099401904   0.015738210200688  -0.014426233258095
   0.000129709433968   0.000729070607420   0.000091336272540   0.000509593124325  -0.000114876001506  -0.000277742638219   0.003593364628836  -0.002022141249569
   0.000260127141369   0.000731261122736   0.001120410178534  -0.000059160025290  -0.000350537070949   0.000050071454134  -0.001358911930891   0.001639842680132
  -0.000245816597477   0.000436005932451  -0.000142314711170   0.000110054837984  -0.000370358568412   0.000709291409056  -0.001882519015045   0.001970932480640
  -0.001402661521971  -0.000649088778728  -0.000550553146126  -0.001115162744517  -0.000603977808993   0.000028273704776  -0.001256431270357   0.003139068965338

  Columns 9 through 15

   0.002004525351092   0.000778794731730   0.000401214562367   0.001062129122933  -0.001533876775809   0.000149160467925   0.004026124721734
   0.000553654743112   0.000625834444467  -0.000029519456860  -0.000034769761504   0.003067287111970   0.002120693469638   0.002048597501565
   0.000058219486888   0.000889850570647  -0.000675351538860  -0.000122752585001   0.001888892148408   0.004886161459059   0.002017661470652
   0.000058029831324   0.001403998556169  -0.000840218555299   0.000393795315309   0.001127560206838   0.001569117486465   0.002212964741286
   0.002296385491293   0.000612035248538   0.000023828312465   0.001322656482533   0.000273964524630   0.004804478242867   0.001802107713850
   0.003792369820698   0.000032505004341   0.000872639131427   0.000315034693695   0.001497250445933   0.001461932190523  -0.001648925301859
   0.000322515621964   0.000853008561505   0.000055461386366   0.000308286705854  -0.000992155766482  -0.001180093725552   0.000175698496199
  -0.003763644039602  -0.000339887768176   0.000526129713761   0.000763242419331   0.002626284079151  -0.000374813624419   0.002575983273996
   1.000364164858882   0.000328791146801  -0.000599434741062   0.000428737625866  -0.001717612181581   0.003335958979285   0.001933615943011
   0.013491600279771   0.999103087654291   0.000370753359440   0.000219164979904  -0.005161012974548   0.006836784857037   0.008802259940427
   0.003948846249448  -0.000162262525927   1.000568276934099  -0.000546370066612  -0.011615901747967   0.011519183232971   0.008575319892059
   0.006893651071019   0.000643812929126   0.000078169119020   0.999511019770776  -0.003341569085735  -0.003900694437288   0.000386629101448
   0.003615064565141  -0.000001056712032   0.000449491327806   0.000748730499525   1.002530733884167   0.000480070280824   0.003780490383340
   0.002362012270299   0.000743152118556   0.000157276586768   0.000101821104187   0.002882804291097   1.000311602728982   0.003718971669788
   0.004125884661175   0.001617641139789   0.000955005815652   0.001763861498291  -0.001695711689924   0.003421284156323   0.999234394525234
```

Fig. 48: Tuned Process Noise Covariance

```
R =

  Columns 1 through 8

   0.676232308960745   0.018855072262165   1.012166257561625   0.592201254902621   0.700340783488738   0.691033882403367   0.019207636172147   0.221498615971171
   0.274559890365457   0.907218112945414   0.295463057653284   0.130489654967001   0.272316681847443   0.493382590054399   0.286428554788286   0.143450813906378
   0.502050925068704   0.014739324162529   1.176752338419093   0.369720711005123   0.111312881797709   0.187729014585307   0.688209546783043   0.621234071592225
   0.149741007690904   0.311683866749145   0.088083040919597   1.097277781751274   0.529560993414412   0.454571756475413   0.240989326632700   0.046190005889164
   0.057842790428626   0.054850770940934   0.674862476623414   0.393657616286936   1.191443793514379   0.178675209344238   0.585128667824469   0.081596899585289
   0.388251770848924   0.794705181497368   0.185031602653400   0.957939285356476   0.051007225750788   1.292078448498887   0.973806130167940   0.916749889341738
   0.219677401311452   0.784387914171348   0.427393405715460   0.644281251911368   0.199978784561713   1.426476085497432   1.039258406960106   0.780000978449114
   0.349663959391884   0.139472054653090   0.036300580148276   0.141224589969566   0.171942741356560   0.536740117378651   0.559395671014758   1.343432221267143
   0.203534144443045   0.435173534495281   0.391270717457908   0.119549967148201   0.036699927386146   0.620438023998016   0.178409291030164   0.143168606541735
   0.298632549738235   0.273963588385521   0.630135347113155   0.655965700297390   0.119436997600481   0.037381321115093   0.435792460481858   0.526527250978878
   0.576780613208339   0.084454277111828   0.239101040259960   0.283049120292705   0.442269755012733   0.403686301979398   0.165052583783299   0.327390529174778
   1.009968568679358   0.115421788926990   0.464594027868084   0.348083788791165   0.064242147894130   0.140703875155664   0.807818057328447   0.745343656785666
   1.433172340464890   0.290499954677428   0.859774324343220   0.170018170528760   0.591097104419468   0.049140933113447   0.477510824686203   0.441140667555682
   0.337138947441235   0.774320233275605   0.220408056113979   0.124739793274991   0.367565383748080   0.014815967568282   0.196552766310478   0.414962684106761
   0.485277653050160   0.071182495001280   0.327815107266789   0.118744308093933   0.151927492029223   0.098843169234291   0.106950192959959   0.262228373407462

  Columns 9 through 15

   0.859724027756296   0.158796000052629   0.200657088536104   0.214109426532220   0.960589821092751   0.603108452307229   0.062041449069659
   0.627985147154725   0.676344556445748   0.151774374664296   0.121900175292077   0.741147888491684   0.064187984640167   0.182134901536862
   0.182711519556500   0.169713863496007   0.024141312610218   0.296422198548243   0.425968036232816   1.367973294611025   0.138284101950092
   0.467653945666265   0.033603219036595   0.209658339260732   0.074276567494596   0.002579568466186   0.222105733809475   0.479826993109759
   0.351238988149507   0.132364433736565   0.130214626267312   0.024742869938034   1.342638222376389   0.709517002733235   0.649497122688525
   0.607704450794897   0.268486108400594   0.048563097532270   0.486368064133656   0.423662295010213   0.459962754733358   0.436708964543684
   0.611081554586343   0.319747881204321   0.045733940783552   0.507737925811401   0.854706182141786   0.450805929865979   0.129262517439127
   0.464444711301374   0.392442792876913   0.017458572182614   0.422041699285586   0.246358964743678   0.230328807527477   0.542224128746864
   0.911936135266288   0.900700027576819   0.754495663907556   0.067866742303561   0.582548389779705   0.046805751121047   0.589160025086435
   0.335510556790553   1.456579444725356   0.149512399964689   0.469308845870713   0.153580323926335   0.341505572009518   0.370555872615870
   0.503242241012834   0.216984114972041   1.812409110805101   0.419048494387835   0.456795515144227   0.109246951591587   0.708328593568087
   0.009269336529232   0.086430944857621   0.443853543305440   0.659852158484865   0.830692324280017   0.254897527612074   0.060380826377767
   0.667920632355884   0.073259386597753   0.355110988228981   0.321462659060728   1.119367080223334   0.006974917187623   0.039022189453141
   0.818670583583004   0.992557497183932   0.080478230830291   0.550549912568505   0.708415372542959   0.765194172270383   0.477729468804715
   0.192325565568848   0.091689417109616   0.054167387467419   0.049326538363458   0.344641660845596   0.562492537329209   1.130902294565699
```

Fig. 49: Tuned Measurement Noise Covariance

More iteration of the algorithm could be run to further improve the results herein however it was deemed that the effectiveness of the MMM EKF was satisfactorily achieved with the current tuning.

After tuning the external IMU the noise and bias unaccounted for in the MMM state equations and calibration routine was significantly reduced. Figures 39 through 41 illustrate the contrast between the accelerations estimated with the EKF and the raw untuned, IMU. Considering how much the signals were washed out previously due to integration drift and the plethora of other conflicts mentioned in Section 7.2 (made apparent in Figures 35- 36) the non-linear optimization approach was successful as Figures 42 and 43 resolve navigational data representative of the true flight trajectory. While the implementation of the tuned process and measurement covariance matrices lead to a more accurate estimate there are still discrepancies with respect ot the predicted trajectory produced by the CUBE autopilot. This observation is made via Figure 45 which represents how well the MMM EKF matches the autopilot. The key factor responsible for the discrepancy is the inconsistent sample rate associated with the external IMU. Significant Integration drift is to be expected with an average sample rate was only 300 Hz. Nonetheless the effectiveness of the MMM EKF is made blatantly obvious in Figure 44 which shows how over estimated the unfiltered tuned IMU navigational estimates are.

7.4 GPS SPOOFING VIA MAVLINK MESSAGING

The motivation for this research was to determine if an external IMU could be used in place of the navigation prediction coming from a CUBE autopilot. The estimated states from the MMM EKF are to be passed to the autopilot as artificial GPS measurements in place of the actual GPS measurements. This implementation can then be utilized in future works during GPS denied environments. To develop such a method by which spoofing the autopilot's GPS with artificial data was possible, SITL simulations were conducted. With the use of MAVLink based messaging the GPS data going to the autopilot was successfully implemented. Appendix A lists the Python script used for passing estimated navigational states to the autopilot as GPS data. This functionality allows the Quadcopter's autopilot to maintain autonomous flight capability without a reliable GPS signal. Figure 6.5, in Section 6, illustrates a ground control station in the simulated environment reading the spoofed GPS data as it is being passed via MAVLink messaging to the vehicles autopilot.

The real implementation of the developed GPS messaging can be run on any Quadcopter build using an aurdupilot base autopilot (i.e., Cube Black) and a companion computer. For the purposes of this research, physical implementation of this script was not possible as the external IMU measurements were post-processed into state estimates. If the state estimates were made available in real time then they could be passed with the GPS_Spoof.py script running on the Raspberry Pi companion computer. This was however functionally verified as discussed above in the SITL simulation

CHAPTER 8

CONCLUSION AND FUTURE WORK

Throughout the course of this research every detail discussed was intended to highlight the renowned empirical steps of the scientific method leading up to a drawn conclusion. After a thorough review of the fundamental components of an inertial navigation system, a deep dive into the state of the art EKF applications for UAVs was made with a slight emphasis on GPS intrusion detection systems. The underlining objective of this research was to determine if strapdown navigation with an external higher quality IMU utilizing the MMM EKF algorithm is effective enough at estimating inertial state information to serve as a suitable replacement for GPS ensuring autonomous flight capability in GPS denied environments. The state estimates are converted into latitude, longitude, and altitude values and passed artificially via GPS Mavlink messages run from the drones' companion computer. To test the hypothesis of whether or not the MMM EKF algorithm along with the external IMU is an effective state estimator the general experimental process was as follows:

- Develop autonomous flight scripts and verify their performance in a virtual environment running SITL simulations.
- Build a Quadcopter with the necessary suite of sensors essential for mission readiness to collect flight test data.
- Validate the EKF MMM algorithm with 1D rectilinear motion first principles.
- Apply non linear optimization to tune EKF algorithm to flight test data.
- Verify CUBE autopilot GPS can be overwritten via Mavlink messaging in SITL simulations.
- Analyze and draw conclusions from state estimator results.

After the flight test were was reduced and the nonlinear optimizer was applied to tune measurement and process noise covariance matrices the inertial state estimation improved drastically. While the tuned filter proved to be effective it was specific to the flight trajectory and was ineffective for additional flights. Despite these limitation artificial GPS data were successfully transmitted to the autopilot via Mavlink messages, as demonstrated in SITL simulations. The error in the predicted states stemmed from several root causes such as bias, scale factor, misalignment, nonlinearities, and integration drift. There are many alternatives that can lead to improvements in future works.

There are three ways to reduce the state estimation error that have the largest impact. From most to least significant: adding additional sensors for measurement data to incorporate into the EKF estimator, increase the sample rate of measurement data acquisition, and to implement a more robust/all encompassing calibration routine. While the purpose of this research was to test if the MMM EKF algorithm (which only relies on the measurements from an IMU) will suffice in adequate navigation predictions, the CUBE autopilot maintains acceptable levels of state estimation with its internal EKFs relying on additional measurements from a GPS and Compass. The most heavily weighted measurement contributing to the CUBE autopilots navigated state is that of the compass. The more accurate heading measurement can be used to filter out the motion in the other unrealistic directions. Other high tech sensors that are newer to the industry have recently been used in other developments as substitute for GPS measurements. This was mentioned in Section 2 in the works of [21] and [23] that introduced the integration of LIDAR based laser scans with a field of view of 270° and proximity sensors capable of measuring the relative distance between neighboring drones.

The Teensy board used to record measurement data from the Analog devices ADIS 16475 was far from perfect, and did not output at a consistent rate or near the max data throughput for which it is capable. Improvements to be made are to implement the data acquisition executable onto the companion machine running autonomous flight mission (RPI) and ensure the max capabilities are being achieved. Establishing a data buffer may be required to output a batch of data registers at 1MHz as per manufacturer's recommendations. The IMU manufacturers included a factory calibration of the internal sensors and built them into the physical outputs of the analog signals.

The only additional calibration routine applied throughout this research was that of a legacy calibration correction provide by previous works. A more intricate calibration process specific to the flight trajectory and application of the IMU would likely improve the error of the state estimator. Using another system of equations to represent the dynamics of the vehicle for which the inertial states are defined by not only the sensor measurements but also sensor bias and scale factor would lead to a more accurate model of the navigated states. Additional error terms are obtainable with a more robust calibration routine which can then be incorporated into the state estimation.

The implementation of spoofing the autopilot GPS in SITL simulations was a proof of concept that can be expanded on in future works. The idea was to inspire future research in machine learning and artificial intelligence with regard to intrusion detection of (GNSS) systems. Applying principles from [25] and [26] a supervised neural network could be trained with known classified GPS data. Eventually the classifier algorithm could run on the Quadcopter's companion computer in real time utilizing the onboard GPS measurements (passed via Mavlink messges to the compaion computer) as inputs and determine if GPS signal is lost, secure, or compromised. In the case of compromised or lost GPS the EKF state estimators based off the external analog devices IMU would take affect and over ride the Autopilots sensed GPS measurements.

To conclude, this research developed and implemented an EKF performing strapdown navigation relying solely on measurements from an external IMU. Autonomous flight scripts were verified via SITL simulations in a LINUX environment and the EKF MMM algorithm was validated against basic first principals. Ultimately experimental results were gathered with flight test of a Quadcopter build that incorporated all the necessary sensors. The state estimator was a limited success but only when tuned for the specific flight trajectory. A more reliable solution for autonomous flight capability in a GPS denied environment must incorporate more sensed measurements and include a better representation of error states. Passing of the state estimates as GPS data has been proven possible.

REFERENCES

[1] G. Welch and G. Bishop, "An introduction to the kalman filter," Tech. Rep. TR 95-041, Department of Computer Science University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2006.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA),: Springer, 2006.

[3] N. S. Nise, *Control Systems Engineering - Sixth Edition.* 111 River Street, Hoboken, NJ 07030-5774: John Wiley & Sons, Inc., 2011.

[4] Dronecode & The Linux Foundation, "Mavlink common message set." `https://mavlink.io/en/messages/common.html`, 2021. Accessed: 06-13-2021.

[5] AruduPilot Dev Team, "Sitl simulator (software in the loop)." `https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html`, 2021. Accessed: 06-14-2021.

[6] AruduPilot Dev Team, "Ardupilot." `https://ardupilot.org/copter/docs/configuring-hardware.html`, 2021. Accessed: 12-01-2021.

[7] G. Panice, S. Luongo, G. Gigante, D. Pascarella, C. D. Benedetto, A. Vozella, and A. Pescapè, "A svm-based detection approach for gps spoofing attacks to uav," in *2017 23rd International Conference on Automation and Computing (ICAC)*, IEEE, pp. 1 – 11, 2017.

[8] D. H. Titterton and J. L. Weston, *Strapdown Inertial Navigation Technology - 2nd Edition.* 1801 Alexander Bell Drive Suite 500 Reston VA 20191-4344 USA: The Institution of Electrical Engineers and The American Institute of Aeronautics and Astronautics, 2004.

[9] N. Ahmed and M. Chen, "Sliding mode control for quadrotor with disturbance observer," *Advances in Mechanical Engineering*, vol. 10, no. 7, pp. 1 – 16, 2018.

[10] T. Madani and A. Benallegue, "Backstepping control for a quadrotor helicopter," in *International Conference on Intelligent Robots and Systems (IRS)*, IEEE, pp. 1 – 7, 2006.

[11] B. Friedland, *Control System Design: An Introduction to State-Space Methods.* Dover Publications, Inc., 31 East 2nd Street, Mineola, N.Y 11501: Dover, 2005.

[12] D. G. LUENBERGER, "Observing the state of a linear system," *IEEE Trans. on Military Electronics*, vol. MIL-8, pp. 74 – 80, April 1964.

[13] R. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME(J. Basic Engineering)*, vol. 82D, no. 1, March 1960.

[14] R. Kalman and R. Bucy, "New results in linear filtering and prediction theory," *Trans. ASME(J. Basic Engineering)*, vol. 83D, no. 1, March 1961.

[15] KROSSBLADE AEROSPACE, "History of quadcopters and other multirotors." `https://www.krossblade.com/history-of-quadcopters-and-multirotors/`, 2021. Accessed: 2021-4-20.

[16] T. Luukkonen, "Modeling and control of quadcopter," Master's thesis, Aalto University School of Science, Espoo, Finland, Aug. 2011.

[17] P. G. Savage, "Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms," *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, vol. 21, no. 1, pp. 19 – 28, 1998.

[18] Q. Quan, *Introduction to Multicopter Design and Control.* 152 Beach Road, 21-01/04 Gateway East, Singapore 189721, Singapore: Springer Nature, 2017.

[19] S. Bijjahalli, R. Sabatini, and A. Gardi, "Advances in intelligent and autonomous navigation systems for small uas," *Progress in Aerospace Sciences*, vol. 115, p. 100617, 2020.

[20] D. Tedaldi, A. Pretto, and E. Menegatti, "A robust and easy to implement method for imu calibration without external equipments," in *International Conference on Robotics and Automation (ICRA)*, IEEE, (Hong Kong), pp. 3042 – 3049, 2014.

[21] G. Mao, S. Drake, and B. D. O. Anderson, "Design of an extended kalman filter for uav localization," in *Information, Decision and Control*, IEEE, (Adelaide, SA, Australia), pp. 224 – 229, 2007.

[22] A. T. Erdem and A. O. Ercan, "Fusing inertial sensor data in an extended kalman filter for 3d camera tracking," *IEEE TRANSACTIONS ON IMAGE PROCESSING*, vol. 24, no. 2, pp. 535–548, 2015.

[23] Y. Tian, K. Liu, K. Ok, L. Tran, D. Allen, N. Roy, and J. P. How, "Search and rescue under the forest canopy using multiple uavs," *The International Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1201 – 1221, 2020.

[24] A. Bachrach, S. Prentice, R. He, and N. Roy, "Range - robust autonomous navigation in gps-denied environments," *Proceedings for 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1096–1097, 2010.

[25] M. P. Arthur, "Detecting signal spoofing and jamming attacks in uav networks using a lightweight ids," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, IEEE, pp. 1 – 5, 2019.

[26] M. R. Manesh, J. Kenney, W. C. Hu, V. K. Devabhaktuni, and N. Kaabouch, "Detection of gps spoofing attacks on unmanned aerial systems," in *2019 16th IEEE Annual Consumer Communications and Networking Conference (CCNC)*, IEEE, pp. 1 – 6, 2019.

[27] C. C. Aggarwal, *Neural Networks and Deep Learning.* IBM T. J. Watson Research Center International Business Machines, Yorktown Heights, NY, USA: Springer, 2018.

[28] MathWorks, "How simulated annealing works." `https://www.mathworks.com/help/gads/how-simulated-annealing-works.html/`, 2022. Accessed: 01-10-2022.

[29] 3D Robotics, "About dronekit." `https://dronekit-python.readthedocs.io/en/latest/about/overview.html`, 2015. Accessed: 06-13-2021.

[30] Dronecode & The Linux Foundation, "Protocol overview." `https://mavlink.io/en/about/overview.html`, 2021. Accessed: 06-13-2021.

[31] AruduPilot Dev Team, "Mandatory hardware configuration." `https://ardupilot.org/ardupilot/index.html`, 2021. Accessed: 06-14-2021.

[32] AruduPilot Dev Team, "Setting up sitl on linux." `https://ardupilot.org/dev/docs/setting-up-sitl-on-linux.html`, 2021. Accessed: 06-14-2021.

[33] AruduPilot Dev Team, "Using sitl." https://ardupilot.org/dev/docs/using-sitl-for-ardupilot-testing.html, 2021. Accessed: 06-14-2021.

[34] Dronecode & The Linux Foundation, "Qgroundcontrol user guide." https://docs.qgroundcontrol.com/master/en/index.html, 2021. Accessed: 06-14-2021.

APPENDIX A

PYTHON SCRIPTS

```
# Sky Seliquini
# MAE 699

"""
This script is intended to utilize dronekit packages and mavlink commands
when necessary
to execute the following

    - connect to AdruCopter
    - take off to a set altitude of 20m
    - go to way point relative to local starting GPS
    - call specific attributes from my_vehicle.py
    - turn off on board GPS after 2 minutes of flight

Then execute simple commands such as printing out attributes and implement
logic to
calculate various parameters.

Maybe use Brian's Data.py as a guid to produce some data files.
"""

# Import relevant libraries
from __future__ import print_function
import time
import math
from dronekit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal, Command
from my_vehicle import MyVehicle # Custom Vehicle class with the attributes
defined

import argparse
parser = argparse.ArgumentParser(description='Connects to SITL on local PC or
to the telemetry of SOLO Drone.')
parser.add_argument('--connect', default="127.0.0.1:14550",
                    help="Vehicle connection target string.")
args = parser.parse_args()

connection_string = args.connect

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready=True,
vehicle_class=MyVehicle)

#################################DEFINE FUNCTIONS REQUIRED TO EXECUTE THE
SCRIPT GOALS##################################
def get_location_metres(original_location, dNorth, dEast):

    """
    Modified original code from Brian Duval to set alt
    Returns a LocationGlobal object containing the latitude/longitude
`dNorth` and `dEast` metres from the
    specified `original_location`. The returned Location has the same `alt`
value
    as `original_location`.

    The function is useful when you want to move the vehicle around
```

```
specifying locations relative to
    the current vehicle position.
    The algorithm is relatively accurate over small distances (10m within
1km) except close to the poles.
    For more information see:
    http://gis.stackexchange.com/questions/2951/algorithm-for-offsetting-a-
latitude-longitude-by-some-amount-of-meters
    """
    earth_radius=6378137.0 #Radius of "spherical" earth

    #Coordinate offsets in radians
    dLat = dNorth/earth_radius
    dLon = dEast/(earth_radius*math.cos(math.pi*original_location.lat/180))

    #New position in decimal degrees
    newlat = original_location.lat + (dLat * 180/math.pi)
    newlon = original_location.lon + (dLon * 180/math.pi)
    return LocationGlobal(newlat, newlon, vehicle.location.global_frame.alt)

############################################################################
###########################################

def download_mission():
    """
    Download the current mission from the vehicle.
    """
    cmds = vehicle.commands
    cmds.download()
    cmds.wait_ready() # wait until download is complete.

############################################################################
###########################################

def Home_Location_Check():
    while not vehicle.home_location:
        cmds= vehicle.commands
        cmds.download()
        cmds.wait_ready()
    print ("Got Home Location")

############################################################################
###########################################

def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """

    print("Basic pre-arm checks")
    # Don't try to arm until autopilot is ready
    while not vehicle.is_armable:
        print(" Waiting for vehicle to initialise...")
        time.sleep(1)

    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    # Confirm vehicle armed before attempting to take off
    while not vehicle.armed:
        print(" Waiting for arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)  # Take off to target altitude

    # Wait until the vehicle reaches a safe height before processing the goto
    #  (otherwise the command after Vehicle.simple_takeoff will execute
    #   immediately).
    while True:
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        # Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude *
0.95:
            print("Reached target altitude")
            break
        time.sleep(1)

###########################################EXECUTE
SCRIPT############################################################

if connection_string == "127.0.0.1:14550":

    arm_and_takeoff(40)

    print("Set default/target airspeed to 3")
    vehicle.airspeed = 3

    print("Going towards waypoint 1...")
    point1 = get_location_metres(vehicle.home_location,21.1663,-56.7703)
    vehicle.simple_goto(point1)

    # sleep so we can see the change in map
    time.sleep(20)

    print("Going towards waypoint 2...")
    point2 = get_location_metres(point1, 132.1986, 53.2401)
    vehicle.simple_goto(point2)

    # sleep so we can see the change in map
    time.sleep(45)

    print("Going towards waypoint 1...")
    vehicle.simple_goto(point1)

    # sleep so we can see the change in map
    time.sleep(45)

    print("Returning to Launch")
    vehicle.mode = VehicleMode("RTL")

    # sleep so we can see the change in map
    time.sleep(40)
```

```python
# Close vehicle object before exiting script
print("Close vehicle object")
vehicle.close()
```

```
# Sky Seliquini
# MAE 699

# Import relevant libraries
from __future__ import print_function
import time
import pandas
import numpy as np
from datetime import datetime
from dronekit import connect, VehicleMode, Vehicle
from my_vehicle import MyVehicle # Custom Vehicle class with the attributes
defined

import argparse
parser = argparse.ArgumentParser(description='Create attributes from MAVLink
messages')
parser.add_argument('--connect', default='127.0.0.1:14551', # this defines
the connection string as the drones address in the simulation
                    help="Vehicle connection target string. If not specified,
SITL automatically started and used")
args = parser.parse_args()
connection_string = args.connect # Connect to a different port so that two
scripts may run simultaneously

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
vehicle = connect(connection_string, wait_ready = True, vehicle_class =
MyVehicle)

while vehicle.armed:
    # define data channel names
    data_labels = np.array(["time_boot_us", "xacc", "yacc", "zacc", "xgyro",
"ygyro", "zgyro", "xmag", "ymag", "zmag", "time_usec", "lat", "lon", "alt"],
dtype=object)
    data = np.empty([1000, 14])

    for i in range(0,999,1):
        if vehicle.location.global_relative_frame.alt > 1:

            # RAW_IMU data
            time_boot_us = vehicle.raw_imu.time_boot_us
            xacc         = vehicle.raw_imu.xacc
            yacc         = vehicle.raw_imu.yacc
            zacc         = vehicle.raw_imu.zacc
            xgyro        = vehicle.raw_imu.xgyro
            ygyro        = vehicle.raw_imu.ygyro
            zgyro        = vehicle.raw_imu.zgyro
            xmag         = vehicle.raw_imu.xmag
            ymag         = vehicle.raw_imu.ymag
            zmag          = vehicle.raw_imu.zmag
            time_usec = vehicle.gps_raw_int.time_usec

            # GPS_RAW_INT data
            lat = vehicle.gps_raw_int.lat
            lon = vehicle.gps_raw_int.lon
            alt = vehicle.gps_raw_int.alt
            # collect data every 5 seconds
            time.sleep(5)

            # populate data array
            data[i] = [time_boot_us, xacc, yacc, zacc, xgyro, ygyro,zgyro,
xmag, ymag, zmag, time_usec, lat, lon, alt]

            # save final i value for if loop
            a = i

        elif vehicle.location.global_relative_frame.alt < 1:
            # deleted the extra rows in the data array, need help from Brian
getting rid of the last row if its possible,
            # would have used "append" if can add as row instead of a column
            data = np.delete(data, a+1, 0)

    # combine data label array with matrix of data
    Data = np.vstack((data_labels, np.asarray(data, object)))

    # write labeled data to excel file
    File_Save_Time = datetime.now()
    File_Name = File_Save_Time.strftime("%m_%d_%Y__%H:%M:%S")
    pandas.DataFrame(Data).to_csv("SITL/flight_test_Data_" + str(File_Name) +
".csv")

    # notify user data acquisition is complete
    print("date record complete")
```

```
# Sky Seliquini
# MAE 699

"""
Example of how to send GPS_INPUT messages to autopilot
"""

# Import relevant libraries
from __future__ import print_function
import time
import math
from pymavlink import mavutil
from dronekit import connect, VehicleMode, LocationGlobalRelative,
LocationGlobal, Command
from my_vehicle import MyVehicle # Custom Vehicle class with the attributes
defined

import argparse
parser = argparse.ArgumentParser(description='Connects to SITL on local PC or
to the telemetry of SOLO Drone.')
parser.add_argument('--connect', default="127.0.0.1:14551",
                    help="Vehicle connection target string.")
args = parser.parse_args()

connection_string = args.connect

# Connect to the Vehicle
print('Connecting to vehicle on: %s' % connection_string)
#master = connect(connection_string, wait_ready=True,
vehicle_class=MyVehicle)

# Create the connection
master = mavutil.mavlink_connection(connection_string)

# Wait a heartbeat before sending commands
# master.wait_heartbeat()

# GPS_TYPE need to be MAV
while True:
    time.sleep(0.2)
    master.mav.gps_input_send(
        1398,  # Timestamp (micros since boot or Unix epoch)
        0,  # ID of the GPS for multiple GPS inputs
        # Flags indicating which fields to ignore (see GPS_INPUT_IGNORE_FLAGS
enum).
        # All other fields must be provided.
        8 | 16 | 32,
        791394000,  # GPS time (milliseconds from start of GPS week)
        0,  # GPS week number
        6,  # 0-1: no fix, 2: 2D fix, 3: 3D fix. 4: 3D with DGPS. 5: 3D with
RTK
        370452380,  # Latitude (WGS84), in degrees * 1E7
        -762978200,  # Longitude (WGS84), in degrees * 1E7
        30100,  # Altitude (AMSL, not WGS84), in m (positive for up)
        0,  # GPS HDOP horizontal dilution of position in m
        0,  # GPS VDOP vertical dilution of position in m
        0,  # GPS velocity in m/s in NORTH direction in earth-fixed NED frame
    0,  # GPS velocity in m/s in EAST direction in earth-fixed NED frame
    0,  # GPS velocity in m/s in DOWN direction in earth-fixed NED frame
    40,  # GPS speed accuracy in m/s
    200,  # GPS horizontal accuracy in m
    200,  # GPS vertical accuracy in m
    10   # Number of satellites visible.
)

print("spoofing GPS...\n")
```

APPENDIX B

MATLAB SCRIPTS

```
clear
close all
clc
% Sky Seliquini
% Strapdown Navigation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COORDINATE SYSTEM
% Z: Up
% X: Forward (North)
% Y: Left    (West)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Inertial Frame
% Rotation
wI=[0,0,0];       %  Inertial Angular Rate [phidot  thetadot  psidot]

% Translation
aI=[0,0,0];       %  Inertial Acceleration [xddot   yddot     zddot]
vI=[0,0,0];       %  Inertial Velocity     [xdot    ydot      zdot]
rI=[0,0,0];       %  Inertial Position     [x       y         z]

% Body Frame
% Rotation
wB=[0,0,0];       %  Body Angular Rate     [p     q     r]
thetaB=[0,0,0];   %  Body Angle            [pInt  qInt  rInt]

% Translation
aB=[0,0,0];       %  Body Acceleration     [udot  vdot  wdot]
vB=[0,0,0];       %  Body Velocity         [u     v     w]

% Prompt user to to determine if using flight data or SITL data
prompt = 'Is the IMU data Flight data, SITL data, or CUBE data (enter as a
string) ''FLIGHT'', ''SITL'', or ''CUBE'': ';
x = input(prompt);

opt_1 = strcmp(x, 'FLIGHT');
opt_2 = strcmp(x, 'SITL') + 1;
opt_3 = strcmp(x, 'CUBE') + 2;
opt_4 = strcmp(x, 'FAKE') + 3;

if opt_1 == 1

    % load experimental IMU data from drones' autonmous flight
    DATA = data_cleaner('datalog_FLIGHT_TEST_Cul-de-sac_2.txt');
    % only look at data up 17000 sample as flight of interest ends
    A = DATA(2828:10816,1:6);
    A(:,1:6) = -1.*A(:,1:6); % adjustment due to IMU being oriented
differently (rightside up)

    % hardcode correction for bias and scale factor s not modeled
    % corrections based on ground data
    load('bias.mat');
    A(:,1) = A(:,1) - bias(1).*ones(length(A(:,1)),1);
    A(:,2) = A(:,2) - bias(2)*ones(length(A(:,2)),1);
    A(:,3) = A(:,3) - bias(3).*ones(length(A(:,3)),1);

    % convert from g to m/s^2
    A(:,1:3) = A(:,1:3).*9.80665;

    % get sample rate of flight data
    t = DATA(2828:10816,8)./1000;
    T_adis = t;

    ind = length(t);
    dt = zeros(ind-1,1);

    for i = 1:ind-1
        dt(i) = t(i+1) - t(i);
    end

elseif opt_2 == 2

    % load simulated IMU data from arducopter SITL autonomous flight
    DATA = process_data('flight_test_Data_10_19_2021__01:49:45.csv');
    A = DATA(:,2:7);

    % convert from mg to m/s^2
    A(:,1:3) = (A(:,1:3)/1000)*9.80665;

    % convert from millirad/s to deg/s
    A(:,4:6) = (A(:,4:6)/1000)*(180/pi);

    % sample rate is 1000Hz in SITL run
    dt = .001;
    dt = ones(length(A),1)*dt;

elseif opt_3 == 3

    % load experimental CUBE data from drones' autonmous flight
    DATA = load('Truth_sec.mat');

    % [Ax Ay Az Gx Gy Gz]
    A = [DATA.Truth.IMU.AccX DATA.Truth.IMU.AccY DATA.Truth.IMU.AccZ...
        DATA.Truth.IMU.GyrX DATA.Truth.IMU.GyrY DATA.Truth.IMU.GyrZ];

    % convert gryo to deg/s
    A(:,4:6) = A(:,4:6).*(180/pi);

    % get sample rate of flight data
    t = DATA.Truth.IMU.Time;

    ind = length(t);
    dt = zeros(ind-1,1);

    for i = 1:ind-1
        dt(i) = t(i+1) - t(i);
    end
```

```
elseif opt_4 == 4

    dt = .01;
    t = 0:dt:100;
    dt = ones(1,length(t)).*dt;

    % 10 seconds positive acceleration
    A1 = ones(1,length(t(1:find(t==20)))).*2;

    % 20 seconds constant acceleration
    A2 = 0*ones(1,length(t(find(t==20.01):(find(t==80)-1))));

    % 10 seconds negative acceleration
    A3 = -A1;

    % scale the accelerations to make them slower
    A_fake = [A1 A2 A3]./10;
    A_fake_noisy = awgn(A_fake,10,'measured');
    % [Ax Ay=0 Az=0 Gx=0 Gy=0 Gz=0]
    A = zeros(length(A_fake),6);
    A(:,1) = A_fake_noisy;

else
    fprintf('\n');
    disp('Input not compatible with Navigation program');
    fprintf('\n');
    return

end

if opt_1 == 1 || opt_2 == 2

    % Accelerometer calibration routine (legacy data from senior design
project)
    % Establish Variables
    ayz = 0.008148007102794; azx = -1.768933828441438e-04; azy =
7.285392573060921e-04;
    bxa = 0.005807379542338; bya = -0.004916903862676;     bza =
0.015227423596253;
    sxa = 0.997086403363496; sya =  0.998478748869561;     sza =
0.998158676892411;

    Ta = [1 -ayz azy;
          0   1 -azx;
          0   0   1;];

    Ka = [sxa 0  0;
          0  sya 0;
          0   0 sza;];

    Ba = [bxa;
          bya;
          bza;];
    % Apply accelerometer calibration to flight data such that accelation
occur
    % about nominal reading
    accel = Ta*Ka*(A(:,1:3)'+Ba);
    A(:,1:3) = accel';

end

if opt_1 == 1

    CAL_dat = data_cleaner('orientation_check.txt');
    % only look at data up 17000 sample as flight of interest ends
    cal = CAL_dat(1:1000,1:6);
    % convert from mg to m/s^2
    cal(:,1:3) = (cal(:,1:3))*-9.80665;

    %  Gravity calibration
    G  = cal(1:1000,1:3); % collect averages of component accelerations
    gx = mean(G(1:1000,1)); % assuming drone is somewhat level and still
    gy = mean(G(1:1000,2)); % for first few seconds
    gz = mean(G(1:1000,3));
    g  = [gx,gy,gz];

else
    %  Gravity calibration
    G  = A(1:50,1:3); % collect averages of component accelerations
    gx = mean(G(1:50,1)); % assuming drone is somewhat level and still
    gy = mean(G(1:50,2)); % for first few seconds
    gz = mean(G(1:50,3));
    g  = [gx,gy,gz];
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%End Calibration%%%%%%%%%%%%%%%%%%%

% EKF Variables
rI_ekf = [0,0,0];
aI_ekf = [0,0,0];
vI_ekf = [0,0,0];
wI_ekf       = [0,0,0];

orenI_ekf    = [0,0,0];
orenI_ekf(2) = -asin(gx/norm(g));
orenI_ekf(1) = asin(gy/(norm(g)*cos(orenI_ekf(2))));
orenI_ekf(3) = 0;


if opt_1 == 1

    % Calculate Gyroscope bias
    % grab the gryo data from the IMU output
    omega = cal(:,4:6).*(pi/180); % convert to radians
    omega = omega';

    [r, ~] = size(omega);
    i = 1;
    for ind = 1:length(omega)
        if norm(omega(:,ind)) < .01 && abs(norm(omega(:,ind)))>0
            AngVel_noise(:,i) = omega(:,ind);
            i = i + 1;
        end
    end

    Ao =
[mean(AngVel_noise(1,:)),mean(AngVel_noise(2,:)),mean(AngVel_noise(3,:))];

else

    % Calculate Gyroscope bias
    % grab the gryo data from the IMU output
    omega = A(:,4:6).*(pi/180); % convert to radians
    omega = omega';

    [r, ~] = size(omega);
    i = 1;

    for ind = 1:length(A)
        if norm(omega(:,ind)) < .01 && abs(norm(omega(:,ind)))>0
            AngVel_noise(:,i) = omega(:,ind);
            i = i + 1;
        end
    end

    if opt_4 == 4
        Ao = [0, 0, 0];
    else
        Ao =
[mean(AngVel_noise(1,:)),mean(AngVel_noise(2,:)),mean(AngVel_noise(3,:))];
    end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kalman Filter
% initial estimates
x_caret_i=0;
z_i = 0;
% priori covariance
P_bar = eye(15,15);

P = eye(15,15);

% Proccess noise
Q = .001*eye(15);

% Measurement noise
R = .001*eye(15);

% A(3,:) = A(3,:) + .04.*ones(1,length(A(3,:)));

t=0;
```

```
for i = 1:length(A)-1

% Read in data
wB   = A(i,4:6).*(pi/180); % wB @t(i)
aB   = A(i,1:3); % aB @t(i)
g_b  = norm(g) *[-sin(orenI_ekf(2)), sin(orenI_ekf(1))*cos(orenI_ekf(2)),
cos(orenI_ekf(1))*cos(orenI_ekf(2))];%Capture Gravity regardless of
orientation

if opt_4 == 4
    a_bb = aB;
else
    a_bb = aB+g_b; % Acce. in Body frame without gravity
end

% Angular Rate Transformation ******* Changing Body angular rate to Inertial
angular rate
T  = Transform(orenI_ekf); % thetaI @t(i)

% apply bias corrections
wB = wB- Ao;

wI = T*wB';% wI @t(i)
wI = wI';

% Linear Acceleration Transformation ******** Changing body acceleration to
% inertial acceleration
R1 = Rotate(orenI_ekf); % orenI_ekf @t(i)A_BB

if opt_4 == 4
    aI = a_bb;
else
    aI = R1*a_bb';
    aI = aI';
end

%  State Defined @ t(i)
%  X(i,:) = [wB,a_bb,wI,orenI_ekf,aI,vI,rI,v1b,V_BI',rI3];
x_bar = [rI_ekf vI_ekf aI_ekf orenI_ekf wI_ekf];

% state jacobian matrix (A in the paper introduction to EKF paper)
F = [1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0 0 0 0;
     0 1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0 0 0;
     0 0 1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0 0;
     0 0 0 1 0 0 dt(i) 0 0 0 0 0 0 0 0;
     0 0 0 0 1 0 0 dt(i) 0 0 0 0 0 0 0;
     0 0 0 0 0 1 0 0 dt(i) 0 0 0 0 0 0;
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1 0 0 dt(i) 0 0;
     0 0 0 0 0 0 0 0 0 0 1 0 0 dt(i) 0;
     0 0 0 0 0 0 0 0 0 0 0 1 0 0 dt(i);
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;];

% Measurement Jacobian Matrix consisting of accel and angular rate
% measurments provided by the IMU
H = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;];



% V: process noise jacobian (W in EKF paper)
V = eye(15);

% W: measurement noise jacobian
W = eye(15);

Z       = [aI wI];
% Kalman Gain
K       = P_bar*H'*inv(H*P_bar*H'+V*R*V');
% new state estimates with measurement
x_caret = x_bar+(K*([zeros(1,6) aI zeros(1,3) wI]-(H*x_bar')')')';
% posteriori covariance
P = (eye(15)-K*H)*P_bar;

%Prediction
P_dot = F*P*F' + W*Q*W';
x_dot = (F*x_caret')'; % B = 0;
P_bar = P_dot;
x_bar = x_dot;

% update x_bar
rI_ekf    = x_bar(1:3);
vI_ekf    = x_bar(4:6);
aI_ekf    = x_bar(7:9);
orenI_ekf = x_bar(10:12);
wI_ekf    = x_bar(13:15);


z_i          = z_i + (Z).*dt(i);
Zi(:,i)      = z_i;
Z_b(:,i)     = Z;
X_BAR(:,i)   = x_bar;
X_CARET(:,i) = x_caret;
```

```
A_BB(:,i)     = a_bb;
AI(:,i)       = aI;
OREN(:,i)     = orenI_ekf*(180/pi);
WI(:,i)       = wI*(180/pi);
WB(:,i)       = wB*(180/pi);
RI_EKF(:,i)   = rI_ekf;
TT(i)         = t;
VI_EKF(:,i)   = vI_ekf;
WI_EKF(:,i)   = wI_ekf*(180/pi);
AI_EKF(:,i)   = aI_ekf;
G_B(:,i)      = g_b;
AB(:,i)       = aB;
t             = dt(i)+t;

end

% convert cube time signal from minutes to seconds
if opt_3 == 3
    TT = TT.*60;
end

if opt_4 == 4

    dt = .01;
    t = 0:dt:100-dt;
    dt = ones(1,length(t)).*dt;


    figure(1)
    plot(t,AI(1,:),'-r')
    hold on
    plot(t,AI_EKF(1,:),'-b')
    hold on
    plot(t,A_fake(1,1:length(t)),'-k',LineWidth=1)
    xlabel('time (s)')
    ylabel('A_{x}   (m/s{^2})')
    title('Acceleration: Basic 1D rectilitnear motion with Guassian white
noise')
    grid on
    legend Ax_{true} Ax_{noisy} Ax_{EKF} Location best

    % calculate Velocity and Position using true acceleration data applying
    % basic kinematic equations
    for idx = 1
        v0 = 0;
        d0 = 0;
        for i = 1:length(dt)
            Vt(idx,i) = v0 + A_fake(idx,i)*dt(i);
            Dt(idx,i) = d0 + v0*dt(i) + .5*A_fake(idx,i)*dt(i)^2;
            v0 = Vt(idx,i);
            d0 = Dt(idx,i);
        end
    end

    % calculate Velocity and Position using EKF acceleration data applying
```

```
        % basic kinematic equations
        for idx = 1
            v0 = 0;
            d0 = 0;
            for i = 1:length(dt)
                Vt_ekf(idx,i) = v0 + AI_EKF(idx,i)*dt(i);
                Dt_ekf(idx,i) = d0 + v0*dt(i) + .5*AI_EKF(idx,i)*dt(i)^2;
                v0 = Vt_ekf(idx,i);
                d0 = Dt_ekf(idx,i);
            end
        end

        figure(2)
        plot(t,VI_EKF(1,:))
        hold on
        plot(t,Vt)
        xlabel('time (s)')
        ylabel('V_{x} (m/s)')
        title('Velocity: Basic 1D rectilitnear motion')
        grid on
        legend Vx_{EKF} Vx_{true} Location best

        figure(3)
        plot(t,RI_EKF(1,:))
        hold on
        plot(t,Dt)
        xlabel('time (s)')
        ylabel('P_{x}(m)')
        title('Position: Basic 1D rectilitnear motion')
        grid on
        legend Rx_{EKF} Rx_{true} Location best

    else
        % integrate filtered and unfiltered IMU accelerations for inertial
        % velocities and positions
        A_xyz = cell(1,2);
        A_xyz{1,1} = AI;
        A_xyz{1,2} = AI_EKF;

        % preallocate memory
        V = cell(1,2);
        Vt = cell(1,2);
        D = cell(1,2);
        Dt = cell(1,2);

        % velocities and positions
        for i = 1:length(A_xyz)
            for j = 1:3
                v0 = 0;
                d0 = 0;
                for k = 1:length(dt)
                    Vt{1,i}(j,k) = v0 + A_xyz{1,i}(j,k)*dt(k);
                    Dt{1,i}(j,k) = d0 + v0*dt(k) + .5*A_xyz{1,i}(j,k)*dt(k)^2;
                    v0 = Vt{1,i}(j,k);
                    d0 = Dt{1,i}(j,k);
                end
            end
        end

        figure(1)
        subplot(3,1,1)
        plot(TT,Dt{1,1}(1,:),'-r')
        hold on
        plot(TT,Dt{1,2}(1,:),'-b')
        title(' X-Position')
        legend('X','X_{ekf}','location','Best')
        xlabel('Time (s)')
        ylabel('X (m)')
        grid on

        subplot(3,1,2)
        plot(TT,Dt{1,1}(2,:),'-r')
        hold on
        plot(TT,Dt{1,2}(2,:),'-b')
        title('Y-Position')
        legend('Y','Y_{ekf}','location','Best')
        xlabel('Time (s)')
        ylabel('Y (m)')
        grid on

        subplot(3,1,3)
        plot(TT,Dt{1,1}(3,:),'-r')
        hold on
        plot(TT,Dt{1,2}(3,:),'-b')
        title('Z-Position')
        legend('Z','Z_{ekf}','location','Best')
        xlabel('Time (s)')
        ylabel('Z (m)')
        grid on

        figure(2)
        subplot(3,1,1)
        plot(TT,Vt{1,1}(1,:),'-r')
        hold on
        plot(TT,Vt{1,2}(1,:),'-b')
        title(' X-Velocity')
        legend('Vx','Vx_{ekf}','location','Best')
        xlabel('Time (s)')
        ylabel('V_{x} (m/s)')
        grid on

        subplot(3,1,2)
        plot(TT,Vt{1,1}(2,:),'-r')
        hold on
        plot(TT,Vt{1,2}(2,:),'-b')
        title('Y-Velocity')
        legend('Vy','Vy_{ekf}','location','Best')
        xlabel('Time (s)')
        ylabel('V_{y} (m/s)')
        grid on
```

```
    subplot(3,1,3)
    plot(TT,Vt{1,1}(3,:),'-r')
    hold on
    plot(TT,Vt{1,2}(3,:),'-b')
    title('Z-Velocity')
    legend('Vz','Vz_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('V_{z} (m/s)')
    grid on

    figure(3)
    plot(TT,AI(1,:))
    hold on
    plot(TT,AI_EKF(1,:))
    title(' X-Acceleration')
    legend('Ax_{IMU}','Ax_{ekf}','location','best')
    xlabel('Time (s)')
    ylabel('A_{x} (m/s{^2})')
    grid on

    figure(4)
    plot(TT,AI(2,:))
    hold on
    plot(TT,AI_EKF(2,:))
    title('Y-Acceleration')
    legend('Ay_{IMU}','Ay_{ekf}','location','best')
    xlabel('Time (s)')
    ylabel('A_{y} (m/s{^2})')
    grid on

    figure(5)
    plot(TT,AI(3,:))
    hold on
    plot(TT,AI_EKF(3,:))
    title('Z-Acceleration')
    legend('Az_{IMU}','Az_{ekf}','location','best')
    xlabel('Time (s)')
    ylabel('A_{z} (m/s{^2})')
    grid on

    % integrate filtered and unfiltered gyroscope data for vehicle
orientation
    W_xyz = cell(1,2);
    W_xyz{1,1} = WI;
    W_xyz{1,2} = WI_EKF;

    % preallocate memory
    W = cell(1,2);
    Wt = cell(1,2);

    % angular position
    for i = 1:length(W_xyz)
        for j = 1:3
            for k = 1:length(dt)
                W{1,i}(j,k) = W_xyz{1,i}(j,k)*dt(k);
            end
        end
    end

    for i = 1:length(W)
        for j= 1:3
            w = 0;
            for k = 1:length(W{1,i}(j,:))
                w = W{1,i}(j,k) + w;
                Wt{1,i}(j,k) = w;
            end
        end
    end

    figure(6)
    subplot(3,1,1)
    plot(TT, Wt{1,1}(1,:),'-r')
    hold on
    plot(TT, Wt{1,2}(1,:),'-b')
    title('Vehicle Orientation')
    legend('\phi','\phi_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('q_{x} (deg)')
    grid on
    subplot(3,1,2)
    plot(TT, Wt{1,1}(2,:),'-r')
    hold on
    plot(TT, Wt{1,2}(2,:),'-b')
    legend('\theta','\theta_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('q_{y} (deg)')
    grid on
    subplot(3,1,3)
    plot(TT, Wt{1,1}(3,:),'-r')
    hold on
    plot(TT, Wt{1,2}(3,:),'-b')
    legend('\psi','\psi_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('q_{z} (deg)')
    grid on

    figure(7)
    subplot(3,1,1)
    plot(TT, WI(1,:), TT, WI_EKF(1,:))
    title('Angular Velocity')
    legend('\omega_{x}_{IMU}','\omega_{x}_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('\omega_{x} (deg/s)')
    grid on
    subplot(3,1,2)
    plot(TT, WI(1,:), TT, WI_EKF(1,:))
    legend('\omega_{y}_{IMU}','\omega_{y}_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('\omega_{y} (deg/s)')
```

```
    grid on
    subplot(3,1,3)
    plot(TT, WI(1,:), TT, WI_EKF(1,:))
    legend('\omega_{z}_{IMU}','\omega_{z}_{ekf}','location','Best')
    xlabel('Time (s)')
    ylabel('\omega_{z} (deg/s)')
    grid on

    % save all matlab figures
    figHandles = findall(0,'Type','figure');

    % Create filename
     fn = strcat(pwd,'\',x,'_EKF_plots');

     % Save first figure
     export_fig(fn, '-pdf', figHandles(1))

     % Loop through figures 2:end
     for i = 2:numel(figHandles)
         export_fig(fn, '-pdf', figHandles(i), '-append')
     end
end
```

```matlab
% Sky Seliquini
% Strapdown Navigation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial Conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COORDINATE SYSTEM
% Z: Up
% X: Forward (North)
% Y: Left    (West)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% initial guess: sky = eye(15);
% x = reshape(sky,1, 225);
close all
clear
clc

% Inertial Frame
% Rotation
wI=[0,0,0];       %  Inertial Angular Rate [phidot  thetadot  psidot]

% Translation
aI=[0,0,0];       %  Inertial Acceleration [xddot   yddot     zddot]
vI=[0,0,0];       %  Inertial Velocity     [xdot    ydot      zdot]
rI=[0,0,0];       %  Inertial Position     [x       y         z]

% Body Frame
% Rotation
wB=[0,0,0];       %  Body Angular Rate     [p     q     r]
thetaB=[0,0,0];   %  Body Angle            [pInt  qInt  rInt]

% Translation
aB=[0,0,0];       %  Body Acceleration     [udot  vdot  wdot]
vB=[0,0,0];       %  Body Velocity         [u     v     w]

% load experimental IMU data from drones' autonmous flight
% flight 2
DATA = data_cleaner('datalog_FLIGHT_TEST_Cul-de-sac_2.txt');
%only look at data associated with test flight
A = DATA(2828:10816,1:6);

% % flight 1
% DATA = data_cleaner('datalog_FLIGHT_TEST_Cul-de-sac_1.txt');
% % only look at data associated with test flight
% A = DATA(806:8629,1:6);

A(:,1:6) = -1.*A(:,1:6); % adjustment due to IMU being oriented differently
(rightside up)

% hardcode correction for bias and scale factor s not modeled
% corrections based on ground data
load('bias.mat');
A(:,1) = A(:,1) - bias(1).*ones(length(A(:,1)),1);
```

```
A(:,2) = A(:,2) - bias(2).*ones(length(A(:,2)),1);
A(:,3) = A(:,3) - bias(3).*ones(length(A(:,3)),1);

% convert from g to m/s^2
A(:,1:3) = A(:,1:3).*9.80665;

% get sample rate of flight data
% flight 2
t = DATA(2828:10816,8)./1000;

% % flight 1
% t = DATA(806:8629,8)./1000;

T_adis = t;

ind = length(t);
dt = zeros(ind-1,1);

for i = 1:ind-1
    dt(i) = t(i+1) - t(i);
end

CAL_dat = data_cleaner('orientation_check.txt');
% only look at data up 17000 sample as flight of interest ends
cal = CAL_dat(1:1000,1:6);
% convert from mg to m/s^2
cal(:,1:3) = (cal(:,1:3))*-9.80665;

%  Gravity calibration
G  = cal(1:1000,1:3); % collect averages of component accelerations
gx = mean(G(1:1000,1)); % assuming drone is somewhat level and still
gy = mean(G(1:1000,2)); % for first few seconds
gz = mean(G(1:1000,3));
g  = [gx,gy,gz];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%End Calibration%%%%%%%%%%%%%%%%%%

% EKF Variables
rI_ekf = [0,0,0];
aI_ekf = [0,0,0];
vI_ekf = [0,0,0];
wI_ekf       = [0,0,0];

orenI_ekf    = [0,0,0];
orenI_ekf(2) = -asin(gx/norm(g));
orenI_ekf(1) = asin(gy/(norm(g)*cos(orenI_ekf(2))));
orenI_ekf(3) = 0;

% Calculate Gyroscope bias
% grab the gryo data from the IMU output
omega = cal(:,4:6).*(pi/180); % convert to radians
omega = omega';

[r, ~] = size(omega);
i = 1;
```

```
for ind = 1:length(omega)
    if norm(omega(:,ind)) < .01 && abs(norm(omega(:,ind)))>0
        AngVel_noise(:,i) = omega(:,ind);
        i = i + 1;
    end
end

Ao =
[mean(AngVel_noise(1,:)),mean(AngVel_noise(2,:)),mean(AngVel_noise(3,:))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kalman Filter
% initial estimates
x_caret_i=0;
z_i = 0;
% priori covariance
P_bar = eye(15,15);
% posteriori covariance
P = eye(15,15);

load('Cov_tuned.mat');
% Proccess noise
Q = reshape(x(226:450),15,15)';

% Measurement noise
R = reshape(x(1:225),15,15)';

% % untuned Covariances
% Q = .001.*eye(15);
% R = .001.*eye(15);

t=0;
for i = 1:length(A)-1

% Read in data
wB   = A(i,4:6).*(pi/180); % wB @t(i)
aB   = A(i,1:3); % aB @t(i)
g_b  = norm(g) *[-sin(orenI_ekf(2)), sin(orenI_ekf(1))*cos(orenI_ekf(2)),
cos(orenI_ekf(1))*cos(orenI_ekf(2))];%Capture Gravity regardless of
orientation


a_bb = aB+g_b; % Acce. in Body frame without gravity


% Angular Rate Transformation ******* Changing Body angular rate to Inertial
angular rate
T  = Transform(orenI_ekf); % thetaI @t(i)

% apply bias corrections
wB = wB- Ao;

if norm(wB)< .01
    wB= [0,0,0];
```

```
end

wI = T*wB';% wI @t(i)
wI = wI';

% Linear Acceleration Transformation ******** Changing body acceleration to
% inertial acceleration
R1 = Rotate(orenI_ekf); % orenI_ekf @t(i)A_BB

aI = R1*a_bb';
aI = aI';

%  State Defined @ t(i)
%  X(i,:) = [wB,a_bb,wI,orenI_ekf,aI,vI,rI,v1b,V_BI',rI3];
x_bar = [rI_ekf vI_ekf aI_ekf orenI_ekf wI_ekf];

% state jacobian matrix (A in the paper introduction to EKF paper)
F = [1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0 0 0;
     0 1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0 0;
     0 0 1 0 0 dt(i) 0 0 (dt(i)^2)/2 0 0 0 0 0 0;
     0 0 0 1 0 0 dt(i)/2 0 0 0 0 0 0 0 0;
     0 0 0 0 1 0 0 dt(i)/2 0 0 0 0 0 0 0;
     0 0 0 0 0 1 0 0 dt(i)/2 0 0 0 0 0 0;
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 1 0 0 dt(i)/2 0 0;
     0 0 0 0 0 0 0 0 0 0 1 0 0 dt(i)/2 0;
     0 0 0 0 0 0 0 0 0 0 0 1 0 0 dt(i)/2;
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;];

% Measurement Jacobian Matrix consisting of accel and angular rate
% measurments provided by the IMU
H = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 1 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 1 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 1 0 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 1 0;
     0 0 0 0 0 0 0 0 0 0 0 0 0 0 1;];



% V: process noise jacobian (W in EKF paper)
V = eye(15);
```

```
% W: measurement noise jacobian
W = eye(15);

Z       = [aI wI];
% Kalman Gain
K       = P_bar*H'*inv(H*P_bar*H'+V*R*V');
% new state estimates with measurement
x_caret = x_bar+(K*([zeros(1,6) aI zeros(1,3) wI]-(H*x_bar')')')';
% updated error covarience
P = (eye(15)-K*H)*P_bar;

%Prediction
P_dot = F*P*F' + W*Q*W';
x_dot = (F*x_caret')'; % B = 0;
P_bar = P_dot;
x_bar = x_dot;

% update x_bar
rI_ekf    = x_bar(1:3);
vI_ekf    = x_bar(4:6);
aI_ekf    = x_bar(7:9);
orenI_ekf = x_bar(10:12);
wI_ekf    = x_bar(13:15);


z_i          = z_i + (Z).*dt(i);
Zi(:,i)      = z_i;
Z_b(:,i)     = Z;
X_BAR(:,i)   = x_bar;
X_CARET(:,i) = x_caret;
A_BB(:,i)    = a_bb;
AI(:,i)      = aI;
OREN(:,i)    = orenI_ekf*(180/pi);
WI(:,i)      = wI*(180/pi);
WB(:,i)      = wB*(180/pi);
RI_EKF(:,i)  = rI_ekf;
TT(i)        = t;
VI_EKF(:,i)  = vI_ekf;
WI_EKF(:,i)  = wI_ekf*(180/pi);
AI_EKF(:,i)  = aI_ekf;
G_B(:,i)     = g_b;
AB(:,i)      = aB;
t            = dt(i)+t;

end

% additional simulated annealing tuning
load('Q2.mat');
A_untuned = [RI_EKF' VI_EKF' AI_EKF' OREN' WI_EKF'];
A_tuned = A_untuned*Q;
RI_EKF = A_tuned(:,1:3)';
VI_EKF = A_tuned(:,4:6)';
AI_EKF = A_tuned(:,7:9)';
OREN = A_tuned(:,10:12)';
```

```matlab
WI_EKF = A_tuned(:,13:15)';

% integrate filtered and unfiltered IMU accelerations for inertial
% velocities and positions
A_xyz = cell(1,2);
A_xyz{1,1} = AI;
A_xyz{1,2} = AI_EKF;

% preallocate memory
V = cell(1,2);
Vt = cell(1,2);
D = cell(1,2);
Dt = cell(1,2);

% velocities and positions
for i = 1:length(A_xyz)
    for j = 1:3
        v0 = 0;
        d0 = 0;
        for k = 1:length(dt)
            Vt{1,i}(j,k) = v0 + A_xyz{1,i}(j,k)*dt(k);
            Dt{1,i}(j,k) = d0 + v0*dt(k) + .5*A_xyz{1,i}(j,k)*dt(k)^2;
            v0 = Vt{1,i}(j,k);
            d0 = Dt{1,i}(j,k);
        end
    end
end

figure(1)
subplot(3,1,1)
plot(TT,Dt{1,1}(1,:),'-b')
hold on
plot(TT,RI_EKF(1,:),'-r')
title(' X-Position')
legend('X','X_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('X (m)')
grid on

subplot(3,1,2)
plot(TT,Dt{1,1}(2,:),'-b')
hold on
plot(TT,RI_EKF(2,:),'-r')
title('Y-Position')
legend('Y','Y_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('Y (m)')
grid on

subplot(3,1,3)
plot(TT,Dt{1,1}(3,:),'-b')
hold on
plot(TT,RI_EKF(3,:),'-r')
title('Z-Position')
legend('Z','Z_{ekf}','location','Best')
```

```
xlabel('Time (s)')
ylabel('Z (m)')
grid on

figure(2)
subplot(3,1,1)
plot(TT,Vt{1,1}(1,:),'-b')
hold on
plot(TT,VI_EKF(1,:),'-r')
title(' X-Velocity')
legend('Vx','Vx_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('V_{x} (m/s)')
grid on

subplot(3,1,2)
plot(TT,Vt{1,1}(2,:),'-b')
hold on
plot(TT,VI_EKF(2,:),'-r')
title('Y-Velocity')
legend('Vy','Vy_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('V_{y} (m/s)')
grid on

subplot(3,1,3)
plot(TT,Vt{1,1}(3,:),'-b')
hold on
plot(TT,VI_EKF(3,:),'-r')
title('Z-Velocity')
legend('Vz','Vz_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('V_{z} (m/s)')
grid on

figure(3)
plot(TT,AI(1,:))
hold on
plot(TT,AI_EKF(1,:))
title(' X-Acceleration')
legend('Ax_{IMU}','Ax_{ekf}','location','best')
xlabel('Time (s)')
ylabel('A_{x} (m/s{^2})')
grid on

figure(4)
plot(TT,AI(2,:))
hold on
plot(TT,AI_EKF(2,:))
title('Y-Acceleration')
legend('Ay_{IMU}','Ay_{ekf}','location','best')
xlabel('Time (s)')
ylabel('A_{y} (m/s{^2})')
grid on
```

```
figure(5)
plot(TT,AI(3,:))
hold on
plot(TT,AI_EKF(3,:))
title('Z-Acceleration')
legend('Az_{IMU}','Az_{ekf}','location','best')
xlabel('Time (s)')
ylabel('A_{z} (m/s{^2})')
grid on

% integrate filtered and unfiltered gyroscope data for vehicle orientation
W_xyz = cell(1,2);
W_xyz{1,1} = WI;
W_xyz{1,2} = WI_EKF;

% preallocate memory
W = cell(1,2);
Wt = cell(1,2);

% angular position
for i = 1:length(W_xyz)
    for j = 1:3
        for k = 1:length(dt)
            W{1,i}(j,k) = W_xyz{1,i}(j,k)*dt(k);
        end
    end
end

for i = 1:length(W)
    for j= 1:3
        w = 0;
        for k = 1:length(W{1,i}(j,:))
            w = W{1,i}(j,k) + w;
            Wt{1,i}(j,k) = w;
        end
    end
end

figure(6)
subplot(3,1,1)
plot(TT, Wt{1,1}(1,:),'-b')
hold on
plot(TT, OREN(1,:),'-r')
title('Vehicle Orientation')
legend('\phi','\phi_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('q_{x} (deg)')
grid on
subplot(3,1,2)
plot(TT, Wt{1,1}(2,:),'-b')
hold on
plot(TT, OREN(2,:),'-r')
legend('\theta','\theta_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('q_{y} (deg)')
```

```
grid on
subplot(3,1,3)
plot(TT, Wt{1,1}(3,:),'-b')
hold on
plot(TT, OREN(3,:),'-r')
legend('\psi','\psi_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('q_{z} (deg)')
grid on

figure(7)
subplot(3,1,1)
plot(TT, WI(1,:), TT, WI_EKF(1,:))
title('Angular Velocity')
legend('\omega_{x}_{IMU}','\omega_{x}_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('\omega_{x} (deg/s)')
grid on
subplot(3,1,2)
plot(TT, WI(1,:), TT, WI_EKF(1,:))
legend('\omega_{y}_{IMU}','\omega_{y}_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('\omega_{y} (deg/s)')
grid on
subplot(3,1,3)
plot(TT, WI(1,:), TT, WI_EKF(1,:))
legend('\omega_{z}_{IMU}','\omega_{z}_{ekf}','location','Best')
xlabel('Time (s)')
ylabel('\omega_{z} (deg/s)')
grid on

%     % save all matlab figures
%     figHandles = findall(0,'Type','figure');
%
%     % Create filename
%      fn = strcat(pwd,'\',x,'_EKF_plots');
%
%      % Save first figure
%      export_fig(fn, '-pdf', figHandles(1))
%
%      % Loop through figures 2:end
%      for i = 2:numel(figHandles)
%          export_fig(fn, '-pdf', figHandles(i), '-append')
%      end
```

APPENDIX C

TEENSY DAQ SCRIPTS

```
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  November 2017 | Updated July 2021
//  Author: Juan Jose Chong <juan.chong@analog.com>
//  Edited: Sky Seliquini- <sseli001@odu.edu>
//  Edited: Rob Stuart - @bornity - <rob@stuart.org>
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  ADIS16475_Teensy_Expanded_Read.ino
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//
//  This Arduino project interfaces with an ADIS16475 using SPI and the
//  accompanying C++ libraries, reads IMU data in LSBs, scales the data, and
//  outputs measurements to a serial debug terminal (PuTTY) via the onboard
//  USB serial port. The Full IMU data set is read, not just the first 20
bytes.
//
//  This project has been tested on a PJRC 32-Bit Teensy 3.6 Development
Board,
//  but should be compatible with any other embedded platform with some
modification.
//
//  Permission is hereby granted, free of charge, to any person obtaining
//  a copy of this software and associated documentation files (the
//  "Software"), to deal in the Software without restriction, including
//  without limitation the rights to use, copy, modify, merge, publish,
//  distribute, sublicense, and/or sell copies of the Software, and to
//  permit persons to whom the Software is furnished to do so, subject to
//  the following conditions:
//
//  The above copyright notice and this permission notice shall be
//  included in all copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
//  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
//  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
//  NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
//  LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
//  WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
//
//  Pinout for a Teensy 3.6 Development Board
//  RST = D6
//  SCK = D13/SCK
//  CS = D10/CS
//  DOUT(MISO) = D12/MISO
//  DIN(MOSI) = D11/MOSI
//  DR = D2
//
////////////////////////////////////////////////////////////////////////////
///////////////////////////

#include <ADIS16475.h>
#include <SPI.h>
#include <SD.h>
```

```
unsigned long myTime;

// SD CARD
// On the Ethernet Shield, CS is pin 4. Note that even if it's not
// used as the CS pin, the hardware CS pin (10 on most Arduino boards,
// 53 on the Mega) must be left as an output or the SD library
// functions will not work.

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
// Teensy audio board: pin 10
// Teensy 3.5 & 3.6 & 4.1 on-board: BUILTIN_SDCARD
// Wiz820+SD board: pin 4
// Teensy 2.0: pin 0
// Teensy++ 2.0: pin 20
const int chipSelect = BUILTIN_SDCARD;


//ADIS16475
// Uncomment to enable debug
//#define DEBUG

// Initialize Variables
// Temporary Data Array
uint16_t *burstData;

// Checksum variable
int16_t burstChecksum = 0;

// Accelerometer
float AXS, AYS, AZS = 0;

// Gyro
float GXS, GYS, GZS = 0;

// Delta Angle
float DAXS, DAYS, DAZS = 0;

// Delta Velocity
float DVXS, DVYS, DVZS = 0;

// Gyro Bias Offset Correction
float GBXS, GBYS, GBZS = 0;

// Accelerometer Bias Offset Correction
float ABXS, ABYS, ABZS = 0;

// Control registers
int MSC = 0;
int FLTR = 0;
int DECR = 0;

// Temperature
float TEMPS = 0;
```

```
// Time Stamp
float TIME = 0;

// Delay counter variable
int printCounter = 0;

// Call ADIS16475 Class
ADIS16475 IMU(10,2,6); // Chip Select, Data Ready, Reset Pin Assignments

void setup()
{
    Serial.begin(115200); // Initialize serial output via USB
    //IMU.configSPI(); // Configure SPI communication //NOTE .configSPI() is
wrong
    IMU.select(); // Configure SPI communication
    delay(500); // Give the part time to start up
    IMU.regWrite(MSC_CTRL, 0xC1);  // Enable Data Ready, set polarity
    IMU.regWrite(FILT_CTRL, 0x04); // Set digital filter
    IMU.regWrite(DEC_RATE, 0x00), // Disable decimation

    // Read the control registers once to print to screen
    MSC = IMU.regRead(MSC_CTRL);
    FLTR = IMU.regRead(FILT_CTRL);
    DECR = IMU.regRead(DEC_RATE);

    attachInterrupt(2, grabData, RISING); // Attach interrupt to pin 2.
Trigger on the rising edge
    //SD Card Initialization
      //while (!Serial) {
     // ; // wait for serial port to connect.
     // }

    Serial.print("Initializing SD card...");

    // see if the card is present and can be initialized:
      if (!SD.begin(chipSelect)) {
        Serial.println("Card failed, or not present");
        while (1) {
        // No SD card, so don't do anything more - stay stuck here
        }
      }
    Serial.println("card initialized.");
}

// Function used to read register values when an ISR is triggered using the
IMU's DataReady output
void grabData()
{
    burstData = {};
    IMU.select(); // Configure SPI before the read. Useful when talking to
multiple SPI devices
    burstData = IMU.wordBurst(); // Read data and insert into array
}

// Function used to scale all acquired data (scaling functions are included
in ADIS16470.cpp)
```

```
void scaleData()
{
    GXS = IMU.gyroScale(*(burstData + 1)); //Scale X Gyro
    GYS = IMU.gyroScale(*(burstData + 2)); //Scale Y Gyro
    GZS = IMU.gyroScale(*(burstData + 3)); //Scale Z Gyro
    AXS = IMU.accelScale(*(burstData + 4)); //Scale X Accel
    AYS = IMU.accelScale(*(burstData + 5)); //Scale Y Accel
    AZS = IMU.accelScale(*(burstData + 6)); //Scale Z Accel
    TEMPS = IMU.tempScale(*(burstData + 7)); //Scale Temp Sensor
    TIME = (*(burstData + 8)); //Time Stamp
    DAXS = IMU.deltaAngleScale(*(burstData + 11)); // Scale X Delta Angle
    DAYS = IMU.deltaAngleScale(*(burstData + 11)); // Scale Y Delta Angle
    DAZS = IMU.deltaAngleScale(*(burstData + 12)); // Scale Z Delta Angle

}

// Main loop. Print data to the serial port. Sensor sampling is performed in
the ISR
void loop()
{
    printCounter ++;
    if (printCounter >= 500) // Delay for writing data to the serial port
    {
      detachInterrupt(2); //Detach interrupt to avoid overwriting data
      scaleData(); // Scale data acquired from the IMU
      burstChecksum = IMU.checksum(burstData); // Calculate checksum based on
data array
      // open the file.

      File dataFile = SD.open("datalog.txt", FILE_WRITE);

      // if the file is available, write to it:
      if (dataFile) {

        // Print Time Stamp data
        dataFile.println(" ");
        //Serial.print(TIME,2);

        // Print scaled gyro data
        //Serial.print(" ");
        dataFile.print(AXS,4);
        dataFile.print(",");
        dataFile.print(AYS,4);
        dataFile.print(",");
        dataFile.print(AZS,4);

        // Print scaled accel data
        dataFile.print(" ,");
        dataFile.print(GXS,4);
        dataFile.print(",");
        dataFile.print(GYS,4);
        dataFile.print(",");
        dataFile.print(GZS,4);

        //Time associated with data registers
        dataFile.print(" ,");
        dataFile.print((*(burstData + 8)));
        // print time since board turned on to serve as clock in milliseconds
        myTime = millis();
        dataFile.print(" ,");
        dataFile.print(myTime);
        delay(1); //sample at 10000Hz
        dataFile.close();
      }
      else {
      // if the file isn't open, pop up an error:
      Serial.println("error opening datalog.txt");
      }

    #ifdef DEBUG
        detachInterrupt(2); //Detach interrupt to avoid overwriting data
        scaleData(); // Scale data acquired from the IMU
        burstChecksum = IMU.checksum(burstData); // Calculate checksum based
on data array

        // Print Time Stamp data
        Serial.println(" ");
        //Serial.print(TIME,2);

        // Print scaled gyro data
        //Serial.print(" ");
        Serial.print(GXS,3);
        Serial.print(",");
        Serial.print(GYS,3);
        Serial.print(",");
        Serial.print(GZS,3);

        // Print scaled accel data
        Serial.print(" ,");
        Serial.print(AXS,4);
        Serial.print(",");
        Serial.print(AYS,4);
        Serial.print(",");
        Serial.print(AZS,4);

        // Print scaled delta angle data
        Serial.print(" ,");
        Serial.print(DAXS,6);
        Serial.print(",");
        Serial.print(DAYS,6);
        Serial.print(",");
        Serial.print(DAZS,6);

        //Time
        Serial.print(" ,");
        Serial.print((*(burstData + 8)));

    #endif
        printCounter = 0;
        attachInterrupt(2, grabData, RISING);
    }
}
```

```
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  November 2017 | Updated July 2021
//  Author: Juan Jose Chong <juan.chong@analog.com>
//  Edited: Rob Stuart - @bornity - <rob@stuart.org>
//  Edited: Sky Seliquini <sseli001@odu.edu>
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  ADIS16475.h
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//
//  This library provides all the functions necessary to interface the
ADIS16475 IMU with a
//  PJRC 32-Bit Teensy 3.6 Development Board. Functions for SPI
configuration, reads and writes,
//  and scaling are included. This library may be used for the entire
ADIS1646X family of devices
//  with some modification.
//
//  Permission is hereby granted, free of charge, to any person obtaining
//  a copy of this software and associated documentation files (the
//  "Software"), to deal in the Software without restriction, including
//  without limitation the rights to use, copy, modify, merge, publish,
//  distribute, sublicense, and/or sell copies of the Software, and to
//  permit persons to whom the Software is furnished to do so, subject to
//  the following conditions:
//
//  The above copyright notice and this permission notice shall be
//  included in all copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
//  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
//  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
//  NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
//  LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
//  WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
//
////////////////////////////////////////////////////////////////////////////
///////////////////////////

#pragma once

#define ADIS16475_h
#include "Arduino.h"
#include <SPI.h>

// User Register Memory Map from Table 8
// Updated July 2021
#define Reservedx00    0x00  //Reserved
#define DIAG_STAT      0x02  //Diagnostic and operational status
#define X_GYRO_LOW     0x04  //X-axis gyroscope output, lower word
#define X_GYRO_OUT     0x06  //X-axis gyroscope output, upper word
#define Y_GYRO_LOW     0x08  //Y-axis gyroscope output, lower word
#define Y_GYRO_OUT     0x0A  //Y-axis gyroscope output, upper word
#define Z_GYRO_LOW     0x0C  //Z-axis gyroscope output, lower word
```

```
#define Z_GYRO_OUT     0x0E  //Z-axis gyroscope output, upper word
#define X_ACCL_LOW     0x10  //X-axis accelerometer output, lower word
#define X_ACCL_OUT     0x12  //X-axis accelerometer output, upper word
#define Y_ACCL_LOW     0x14  //Y-axis accelerometer output, lower word
#define Y_ACCL_OUT     0x16  //Y-axis accelerometer output, upper word
#define Z_ACCL_LOW     0x18  //Z-axis accelerometer output, lower word
#define Z_ACCL_OUT     0x1A  //Z-axis accelerometer output, upper word
#define TEMP_OUT       0x1C  //Temperature output (internal, not calibrated)
#define TIME_STAMP     0x1E  //PPS mode time stamp
#define Reservedx20   0x20  //Reserved
#define DATA_CNTR     0x22  //New Data Counter
#define X_DELTANG_LOW  0x24  //X-axis delta angle output, lower word
#define X_DELTANG_OUT  0x26  //X-axis delta angle output, upper word
#define Y_DELTANG_LOW  0x28  //Y-axis delta angle output, lower word
#define Y_DELTANG_OUT  0x2A  //Y-axis delta angle output, upper word
#define Z_DELTANG_LOW  0x2C  //Z-axis delta angle output, lower word
#define Z_DELTANG_OUT  0x2E  //Z-axis delta angle output, upper word
#define X_DELTVEL_LOW  0x30  //X-axis delta velocity output, lower word
#define X_DELTVEL_OUT  0x32  //X-axis delta velocity output, upper word
#define Y_DELTVEL_LOW  0x34  //Y-axis delta velocity output, lower word
#define Y_DELTVEL_OUT  0x36  //Y-axis delta velocity output, upper word
#define Z_DELTVEL_LOW  0x38  //Z-axis delta velocity output, lower word
#define Z_DELTVEL_OUT  0x3A  //Z-axis delta velocity output, upper word
#define Reservedx3C   0x3C  //Reserved
#define Reservedx3E   0x3E  //Reserved
#define XG_BIAS_LOW       0x40  //X-axis gyroscope bias offset correction,
lower word
#define XG_BIAS_HIGH   0x42  //X-axis gyroscope bias offset correction, upper
word
#define YG_BIAS_LOW       0x44  //Y-axis gyroscope bias offset correction,
lower word
#define YG_BIAS_HIGH   0x46  //Y-axis gyroscope bias offset correction, upper
word
#define ZG_BIAS_LOW       0x48  //Z-axis gyroscope bias offset correction,
lower word
#define ZG_BIAS_HIGH   0x4A  //Z-axis gyroscope bias offset correction, upper
word
#define XA_BIAS_LOW       0x4C  //X-axis accelerometer bias offset
correction, lower word
#define XA_BIAS_HIGH   0x4E  //X-axis accelerometer bias offset correction,
upper word
#define YA_BIAS_LOW       0x50  //Y-axis accelerometer bias offset
correction, lower word
#define YA_BIAS_HIGH   0x52  //Y-axis accelerometer bias offset correction,
upper word
#define ZA_BIAS_LOW       0x54  //Z-axis accelerometer bias offset
correction, lower word
#define ZA_BIAS_HIGH   0x56  //Z-axis accelerometer bias offset correction,
upper word
#define Reservedx58   0x58  //Reserved
#define Reservedx5A   0x5A  //Reserved
#define FILT_CTRL      0x5C  //Control, Bartlett window FIR filter
#define RANG_MDL      0x5E  //Measurement range (model specific) identifier
#define MSC_CTRL       0x60  //Control, input/output and other miscellaneous
options
#define UP_SCALE       0x62  //Control, scale factor for input clock, pulse
per second (PPS) mode
```

```
#define DEC_RATE       0x64  //Control, decimation filter (output data rate)
#define NULL_CFG       0x66  //Control, bias estimation period (Default =
0x070A)
#define GLOB_CMD       0x68  //Control, global commands
#define Reservedx6A   0x6A  //Reserved
#define FIRM_REV       0x6C  //Firmware revision
#define FIRM_DM           0x6E  //Firmware revision date, month and day
#define FIRM_Y        0x70  //Firmware revision date, year
#define PROD_ID           0x72  //Product identification
#define SERIAL_NUM    0x74  //Serial number (relative to assembly lot)
#define USER_SCR1      0x76  //User scratch register 1
#define USER_SCR2      0x78  //User scratch register 2
#define USER_SCR3      0x7A  //User scratch register 3
#define FLSHCNT_LOW   0x7C  //Flash update count, lower word
#define FLSHCNT_HIGH  0x7E  //Flash update count, upper word


// ADIS16475 class definition
class ADIS16475 {

public:
  // Constructor with configurable CS, data ready, and HW reset pins

  // ADIS16475(int CS, int DR, int RST, int MOSI, int MISO, int CLK);
  ADIS16475(int CS, int DR, int RST);

  // Destructor
  ~ADIS16475();

  // Performs hardware reset by sending pin 8 low on the DUT for n
milliseconds
  int resetDUT(uint8_t ms);

  // Sets SPI bit order, clock divider, and data mode and sets CS chip to
LOW.
  int select();

  // Disables SPI bus and sets CS chip to HIGH.
  int deselect();

  // Read single register from sensor
  int16_t regRead(uint8_t regAddr);

  // Write register
  int regWrite(uint8_t regAddr, int16_t regData);

  // Read sensor data using a burst read. Returns bits
  uint8_t *byteBurst(void);

  // Read sensor data using a burst read. Returns bytes
  uint16_t *wordBurst(void);

  // Calculate checksum
  int16_t checksum(uint16_t * burstArray);

  // Scale accelerator data
  float accelScale(int16_t sensorData);
  // Scale gyro data
  float gyroScale(int16_t sensorData);

  // Scale temperature data
  float tempScale(int16_t sensorData);

  // Scale delta angle data
  float deltaAngleScale(int16_t sensorData);

  // Scale delta velocity
  float deltaVelocityScale(int16_t sensorData);

  // Scale delta velocity
  float timeStamp(int16_t sensorData);

private:
  // Variables to store hardware pin assignments
  int _CS;
  int _DR;
  int _RST;
  int _stall = 20;

};
```

```
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  November 2017
//  Author: Juan Jose Chong <juan.chong@analog.com>
//  Edited: Rob Stuart - @bornity - <rob@stuart.org>
//  Edited: Sky Seliquini- <sseli001@odu.edu>
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//  ADIS16475.cpp
////////////////////////////////////////////////////////////////////////////
///////////////////////////
//
//  This library provides all the functions necessary to interface the
ADIS16475 IMU with a
//  PJRC 32-Bit Teensy 3.2 Development Board. Functions for SPI
configuration, reads and writes,
//  and scaling are included. This library may be used for the entire
ADIS1646X family of devices
//  with some modification.
//
//  Permission is hereby granted, free of charge, to any person obtaining
//  a copy of this software and associated documentation files (the
//  "Software"), to deal in the Software without restriction, including
//  without limitation the rights to use, copy, modify, merge, publish,
//  distribute, sublicense, and/or sell copies of the Software, and to
//  permit persons to whom the Software is furnished to do so, subject to
//  the following conditions:
//
//  The above copyright notice and this permission notice shall be
//  included in all copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
//  EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
//  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
//  NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
//  LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
//  WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
//
////////////////////////////////////////////////////////////////////////////
///////////////////////////

#include "ADIS16475.h"

////////////////////////////////////////////////////////////////////////////
// Constructor with configurable CS, DR, and RST
////////////////////////////////////////////////////////////////////////////
// CS - Chip select pin
// DR - DR output pin for data ready
// RST - Hardware reset pin
////////////////////////////////////////////////////////////////////////////
ADIS16475::ADIS16475(int CS, int DR, int RST) {
  _CS = CS;
  _DR = DR;
  _RST = RST;
  // Initialize SPI
  SPI.begin();
  // Set default pin states
  pinMode(_CS, OUTPUT); // Set CS pin to be an output
  pinMode(_DR, INPUT); // Set DR pin to be an input
  pinMode(_RST, OUTPUT); // Set RST pin to be an output
  digitalWrite(_CS, HIGH); // Initialize CS pin to be high
  digitalWrite(_RST, HIGH); // Initialize RST pin to be high
}

////////////////////////////////////////////////////////////////////////////
// Destructor
////////////////////////////////////////////////////////////////////////////
ADIS16475::~ADIS16475() {
}

////////////////////////////////////////////////////////////////////////////
// Performs a hardware reset by setting _RST pin low for delay (in ms).
// Returns 1 when complete.
////////////////////////////////////////////////////////////////////////////
int ADIS16475::resetDUT(uint8_t ms) {
  digitalWrite(_RST, LOW);
  delay(ms);
  digitalWrite(_RST, HIGH);
  delay(ms);
  return(1);
}

////////////////////////////////////////////////////////////////////////////
// Selects the ADIS16475 for read/write operations.
// Sets SPI bit order, clock divider, and data mode.
// Also sets chip select to LOW.
// This function is useful when there are multiple SPI devices
// using different settings.
// Returns 1 when complete.
////////////////////////////////////////////////////////////////////////////
int ADIS16475::select() {
  SPISettings IMUSettings(1000000, MSBFIRST, SPI_MODE3);
  SPI.beginTransaction(IMUSettings);
  digitalWrite(_CS, LOW); // Set CS low to enable device
  return (1);
}

////////////////////////////////////////////////////////////////////////////
// Deselects the ADIS16475 for read/write operations.
// Frees up the SPi bus for other devices.
// Also sets chip select to HIGH.
// Returns 1 when complete.
////////////////////////////////////////////////////////////////////////////
int ADIS16475::deselect() {
  SPI.endTransaction();
  digitalWrite(_CS, HIGH); // Set CS high to disable device
  return (1);
}

////////////////////////////////////////////////////////////////////////////
///////////////
// Reads two bytes (one word) in two sequential registers over SPI
// Returns an (int) signed 16 bit 2's complement number
```

```
////////////////////////////////////////////////////////////////////////////
///////////////
// regAddr - address of register to be read
////////////////////////////////////////////////////////////////////////////
///////////////
int16_t ADIS16475::regRead(uint8_t regAddr) {
//Read registers using SPI

  // Write register address to be read
  select();              // select the device
  SPI.transfer(regAddr); // Write address over SPI bus
  SPI.transfer(0x00); // Write 0x00 to the SPI bus fill the 16 bit
transaction requirement
  deselect();            // deselect the device

  delayMicroseconds(_stall); // Delay to not violate read rate

  // Read data from requested register
  select();              // select the device
  uint8_t _msbData = SPI.transfer(0x00); // Send (0x00) and place upper byte
into variable
  uint8_t _lsbData = SPI.transfer(0x00); // Send (0x00) and place lower byte
into variable
  deselect();            // deselect the device

  delayMicroseconds(_stall); // Delay to not violate read rate

  int16_t _dataOut = (_msbData << 8) | (_lsbData & 0xFF); // Concatenate
upper and lower bytes
  // Shift MSB data left by 8 bits, mask LSB data with 0xFF, and OR both
bits.

  return(_dataOut);
}

///////////////////////////////////////////////////////////////////////////
// Writes one byte of data to the specified register over SPI.
// Returns 1 when complete.
///////////////////////////////////////////////////////////////////////////
// regAddr - address of register to be written
// regData - data to be written to the register
///////////////////////////////////////////////////////////////////////////
int ADIS16475::regWrite(uint8_t regAddr, int16_t regData) {

  // Write register address and data
  uint16_t addr = (((regAddr & 0x7F) | 0x80) << 8); // Toggle sign bit, and
check that the address is 8 bits
  uint16_t lowWord = (addr | (regData & 0xFF)); // OR Register address (A)
with data(D) (AADD)
  uint16_t highWord = ((addr | 0x100) | ((regData >> 8) & 0xFF)); // OR
Register address with data and increment address

  // Split words into chars
  uint8_t highBytehighWord = (highWord >> 8);
  uint8_t lowBytehighWord = (highWord & 0xFF);
  uint8_t highBytelowWord = (lowWord >> 8);
  uint8_t lowBytelowWord = (lowWord & 0xFF);
  // Write highWord to SPI bus
  select();               // select the device
  SPI.transfer(highBytelowWord); // Write high byte from low word to SPI bus
  SPI.transfer(lowBytelowWord); // Write low byte from low word to SPI bus
  deselect();             // deselect the device

  delayMicroseconds(_stall);; // Delay to not violate read rate

  // Write lowWord to SPI bus
  select();               // select the device
  SPI.transfer(highBytehighWord); // Write high byte from high word to SPI
bus
  SPI.transfer(lowBytehighWord); // Write low byte from high word to SPI bus
  deselect();             // deselect the device

  delayMicroseconds(_stall);; // Delay to not violate read rate

  return(1);
}


uint8_t *ADIS16475::byteBurst(void) {

  static uint8_t burstdata[32];

  // Trigger Burst Read
  select(); // select the device
  SPI.transfer(0x68);
  SPI.transfer(0x00);

  // Read Burst Data
  burstdata[0] = SPI.transfer(0x00); //DIAG_STAT
  burstdata[1] = SPI.transfer(0x00);
  burstdata[2] = SPI.transfer(0x00); //XGYRO_OUT
  burstdata[3] = SPI.transfer(0x00);
  burstdata[4] = SPI.transfer(0x00); //YGYRO_OUT
  burstdata[5] = SPI.transfer(0x00);
  burstdata[6] = SPI.transfer(0x00); //ZGYRO_OUT
  burstdata[7] = SPI.transfer(0x00);
  burstdata[8] = SPI.transfer(0x00); //XACCEL_OUT
  burstdata[9] = SPI.transfer(0x00);
  burstdata[10] = SPI.transfer(0x00); //YACCEL_OUT
  burstdata[11] = SPI.transfer(0x00);
  burstdata[12] = SPI.transfer(0x00); //ZACCEL_OUT
  burstdata[13] = SPI.transfer(0x00);
  burstdata[14] = SPI.transfer(0x00); //TEMP_OUT
  burstdata[15] = SPI.transfer(0x00);
  burstdata[16] = SPI.transfer(0x00); //TIME_STMP
  burstdata[17] = SPI.transfer(0x00);
  burstdata[18] = SPI.transfer(0x00); //RESERVED
  burstdata[19] = SPI.transfer(0x00);
  burstdata[20] = SPI.transfer(0x00); //XDELTA_ANGLE
  burstdata[21] = SPI.transfer(0x00);
  burstdata[22] = SPI.transfer(0x00); //YDELTA_ANGLE
  burstdata[23] = SPI.transfer(0x00);
  burstdata[24] = SPI.transfer(0x00); //ZDELTA_ANGLE
  burstdata[25] = SPI.transfer(0x00);
  burstdata[26] = SPI.transfer(0x00); //XDELTA_VELOCITY
  burstdata[27] = SPI.transfer(0x00);
  burstdata[28] = SPI.transfer(0x00); //YDELTA_VELOCITY
  burstdata[29] = SPI.transfer(0x00);
  burstdata[30] = SPI.transfer(0x00); //ZDELTA_VELOCITY
  burstdata[31] = SPI.transfer(0x00);
  /*
  burstdata[32] = SPI.transfer(0x00); //XGYRO_BIAS
  burstdata[33] = SPI.transfer(0x00);
  burstdata[34] = SPI.transfer(0x00); //YDGYRO_BIAS
  burstdata[35] = SPI.transfer(0x00);
  burstdata[36] = SPI.transfer(0x00); //ZGYRO_BIAS
  burstdata[37] = SPI.transfer(0x00);
  burstdata[38] = SPI.transfer(0x00); //XACCEL_BIAS
  burstdata[39] = SPI.transfer(0x00);
  burstdata[40] = SPI.transfer(0x00); //YACCEL_BIAS
  burstdata[41] = SPI.transfer(0x00);
  burstdata[42] = SPI.transfer(0x00); //ZACCEL_BIAS
  burstdata[43] = SPI.transfer(0x00);
  burstdata[44] = SPI.transfer(0x00); //
  burstdata[45] = SPI.transfer(0x00);
  */
  deselect(); // deselect the device

  return burstdata;
}

uint16_t *ADIS16475::wordBurst(void) {

  static uint16_t burstwords[17];

  // Trigger Burst Read
  select(); // select the device
  SPI.transfer(0x68);
  SPI.transfer(0x00);

  // Read Burst Data
  burstwords[0] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//DIAG_STAT
  burstwords[1] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//XGYRO
  burstwords[2] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//YGYRO
  burstwords[3] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//ZGYRO
  burstwords[4] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//XACCEL
  burstwords[5] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//YACCEL
  burstwords[6] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//ZACCEL
  burstwords[7] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//TEMP_OUT
  burstwords[8] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//TIME_STMP
  burstwords[9] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
```

```
//CHECKSUM1
  burstwords[10] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//DATA_CNTR
  burstwords[11] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//XDELTA_ANGLE
  burstwords[12] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//YDELTA_ANGLE
  burstwords[13] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//ZDELTA_ANGLE
  burstwords[14] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//XDELTA_VELOCITY
  burstwords[15] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//YDELTA_VELOCITY
  burstwords[16] = ((SPI.transfer(0x00) << 8) | (SPI.transfer(0x00) & 0xFF));
//ZDELTA_VELOCITY

  deselect();  // deselect the device

  return burstwords;
}

////////////////////////////////////////////////////////////////////////////
// Calculates checksum based on burst data.
// Returns the calculated checksum.
////////////////////////////////////////////////////////////////////////////
// *burstArray - array of burst data
// return - (int16_t) signed calculated checksum
////////////////////////////////////////////////////////////////////////////
int16_t ADIS16475::checksum(uint16_t * burstArray) {
  int16_t s = 0;
  for (int i = 0; i < 9; i++) // Checksum value is not part of the sum!!
  {
      s += (burstArray[i] & 0xFF); // Count lower byte
      s += ((burstArray[i] >> 8) & 0xFF); // Count upper byte
  }

  return s;
}

/////////////////////////////////////////////////////////////////////////////
////////////
// Converts accelerometer data output from the regRead() function
// Returns (float) signed/scaled accelerometer in g's
/////////////////////////////////////////////////////////////////////////////
////////////
// sensorData - data output from regRead()
/////////////////////////////////////////////////////////////////////////////
////////////
float ADIS16475::accelScale(int16_t sensorData)
{
  float finalData = sensorData * 0.00025; // Multiply by accel sensitivity
(0.00025g/LSB)
  return finalData;
}

/////////////////////////////////////////////////////////////////////////////
////////////////
```

```
// Converts gyro data output from the regRead() function
// Returns (float) signed/scaled gyro in degrees/sec
////////////////////////////////////////////////////////////////////////////
////////////////
// sensorData - data output from regRead()
////////////////////////////////////////////////////////////////////////////
////////////
float ADIS16475::gyroScale(int16_t sensorData)
{
  float finalData = sensorData * 0.025; // Multiply by gyro sensitivity
(0.025 deg/LSB)
  return finalData;
}

////////////////////////////////////////////////////////////////////////////
////////////////
// Converts temperature data output from the regRead() function
// Returns (float) signed/scaled temperature in degrees Celcius
////////////////////////////////////////////////////////////////////////////
////////////////
// sensorData - data output from regRead()
////////////////////////////////////////////////////////////////////////////
////////////
float ADIS16475::tempScale(int16_t sensorData)
{
  float finalData = (sensorData * 0.1); // Multiply by temperature scale (0.1
deg C/LSB)
  return finalData;
}

////////////////////////////////////////////////////////////////////////////
////////////////
// Converts integrated angle data output from the regRead() function
// Returns (float) signed/scaled delta angle in degrees
////////////////////////////////////////////////////////////////////////////
////////////////
// sensorData - data output from regRead()
////////////////////////////////////////////////////////////////////////////
////////////
float ADIS16475::deltaAngleScale(int16_t sensorData)
{
  float finalData = sensorData * 0.0082; // Multiply by delta angle scale
(0.0082 degrees/LSB)
  return finalData;
}

////////////////////////////////////////////////////////////////////////////
////////////////
// Converts integrated velocity data output from the regRead() function
// Returns (float) signed/scaled delta velocity in m/sec
////////////////////////////////////////////////////////////////////////////
////////////////
// sensorData - data output from regRead()
////////////////////////////////////////////////////////////////////////////
////////////
float ADIS16475::deltaVelocityScale(int16_t sensorData)
{
  float finalData = sensorData * 0.003052; // Multiply by velocity scale
(0.01221 m/sec/LSB)
  return finalData;
}

////////////////////////////////////////////////////////////////////////////
////////////////
// Converts integrated velocity data output from the regRead() function
// Returns (float) signed/scaled delta velocity in m/sec
////////////////////////////////////////////////////////////////////////////
////////////////
// sensorData - data output from regRead()
////////////////////////////////////////////////////////////////////////////
////////////

float ADIS16475::timeStamp(int16_t sensorData)
{
  float finalData = sensorData; //Convert Time Stamp Data
  return finalData;
}
```

APPENDIX D

HARDWARE PINOUT

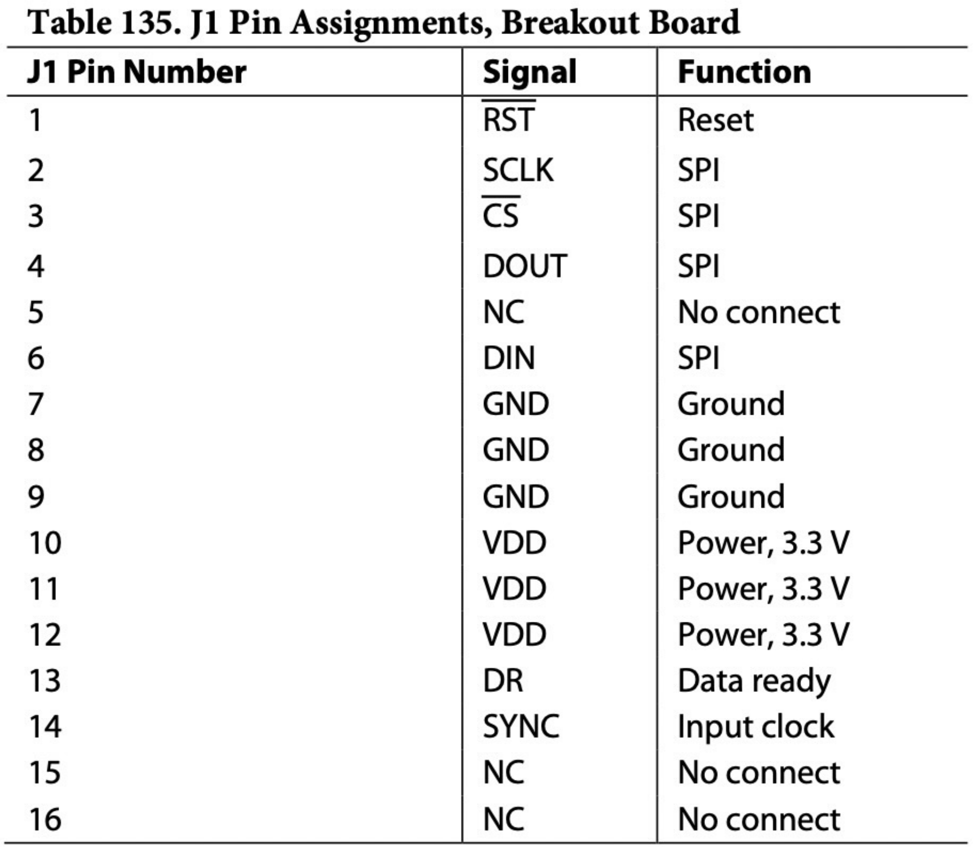


Table 135. J1 Pin Assignments, Breakout Board

J1 Pin Number	Signal	Function
1	$\overline{\text{RST}}$	Reset
2	SCLK	SPI
3	$\overline{\text{CS}}$	SPI
4	DOUT	SPI
5	NC	No connect
6	DIN	SPI
7	GND	Ground
8	GND	Ground
9	GND	Ground
10	VDD	Power, 3.3 V
11	VDD	Power, 3.3 V
12	VDD	Power, 3.3 V
13	DR	Data ready
14	SYNC	Input clock
15	NC	No connect
16	NC	No connect

Fig. 50: Analog Devices ADIS 16475 Pinout

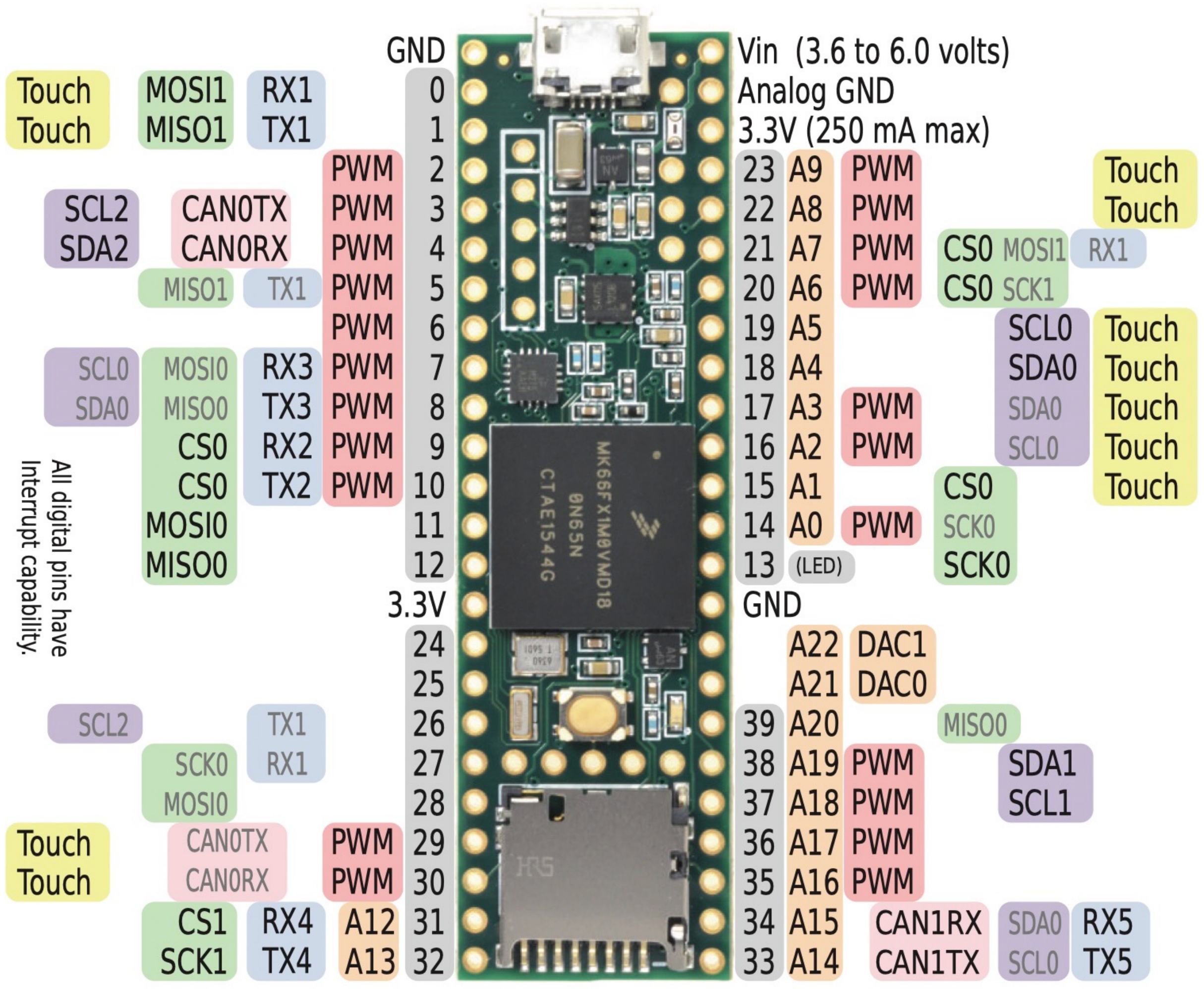


Fig. 51: Teensy 3.6 with Micro SD Pinout

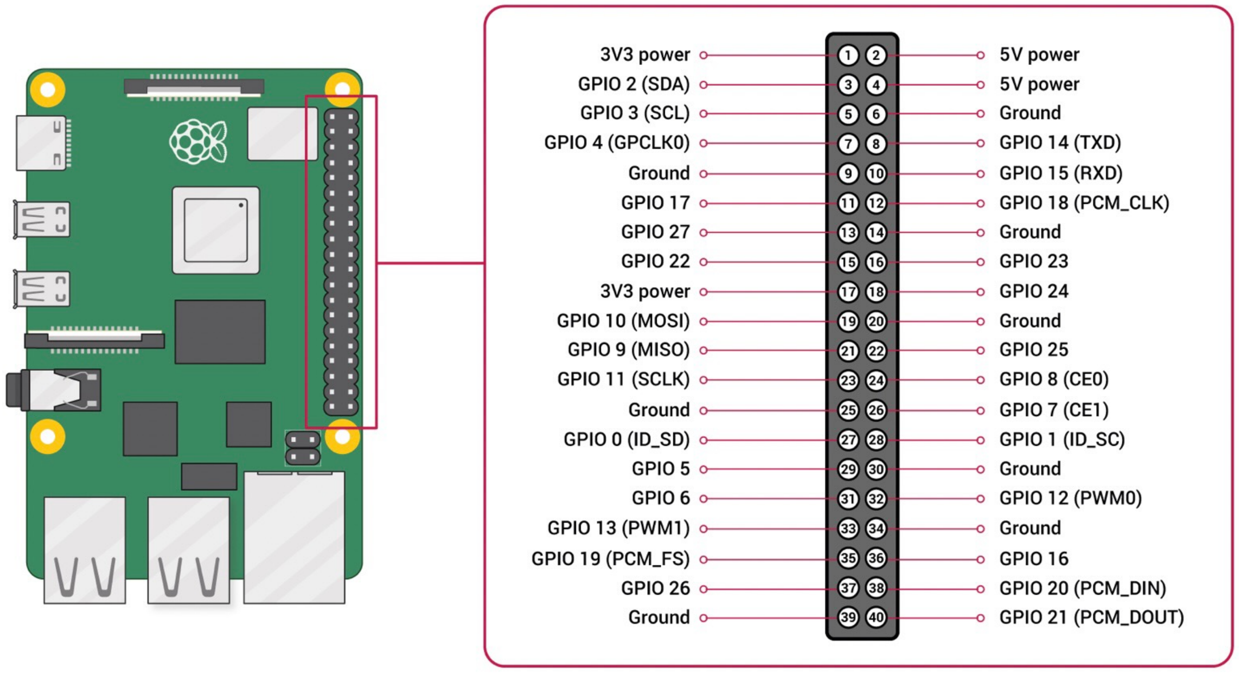


Fig. 52: Raspberry Pi 3B+ Pinout

VITA

Sky Seliquini was born in Rhein Main, Germany on June 7th, 1993. Growing up in a military family, he lived all around the United States and spent the majority of his childhood in Rome, Italy. Prior to attending Old Dominion University he attended Florida Institute of Technology, Melbourne, Florida, where he earned a Bachelors of Science in Aerospace Engineering with a minor in Computational Mathematics in 2015. Immediately after completing his undergraduate degree he started working as a Product Development engineer at Embraer working on aftermarket modifications to executive jets. From 2017 to 2021 he worked full time at NASA Langley's Transonic Dynamics Tunnel as a Wind Tunnel Test Engineer while continuing the pursuit of his education in his free time. Currently Sky is a Principle Guidance Navigation and Control Engineer at Northrop Grumman in Roy, Utah. He lives in Layton, Utah.