SENSOR FUSION TO IMPROVE STATE ESTIMATE ACCURACY USING MULTIPLE INERTIAL MEASUREMENT UNITS AND IT'S APPLICATION TO WIND ESTIMATION

By

UJJVAL NIRMALKUMAR PATEL

Bachelor of Science in Aerospace Engineering
Oklahoma State University
Stillwater, OK
2019

Bachelor of Science in Mechanical Engineering
Oklahoma State University
Stillwater, OK
2019

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2021

SENSOR FUSION TO IMPROVE STATE ESTIMATE ACCURACY
USING MULTIPLE INERTIAL MEASUREMENT UNITS
AND IT'S APPLICATION TO WIND ESTIMATION

Thesis Approved:

Dr. Imraan Faruque

Thesis Advisor

Dr. He Bai

Dr. Jamey Jacob

ACKNOWLEDGMENTS

This thesis would not have been possible without the expertise and unwavering support of my advisor, Dr. Imraan Faruque. I will be forever indebted and thankful for encouragement throughout my time at OSU. He has been a very patient, selfless and ideal advisor, and I highly respect his valuable guidance. I also want to extend my appreciation to my committee members, Dr. He Bai and Dr. Jamey Jacob, for all their help for classes and reviewing my thesis and providing their valuable feedback and time.

I am thankful to my friends and colleagues mainly, I would like to mention Brandon White, Nick Rozell, Kyle Hickman, and Scott Weekley for always being helpful and supportive.

I would also especially like to mention Jayesh Yevale and Kushal Shah who have been like a family far from home and always willing to drag me away from work to keep me living life.

Finally, I must express my very profound gratitude to my family and especially my parents Mr. Nirmal Patel and Ms. Manisha Patel, and to my beloved Dr. Ayushi Naik and oli for providing me with unfailing support, love and continuous encouragement throughout my years. All my accomplishments would not have been possible without them. Thank you.

This work was supported in part by the National Aeronautics and Space Administration under Grant 80NSSC20M0162, University Leadership Initiative.

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: UJJVAL NIRMALKUMAR PATEL

Date of Degree: DECEMBER, 2021

Title of Study: SENSOR FUSION TO IMPROVE STATE ESTIMATE ACCURACY USING MULTIPLE INERTIAL MEASUREMENT UNITS AND IT'S APPLICATION TO WIND ESTIMATION

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: On-board state estimation is a persistent challenge to fielding unmanned aerial systems (UAS), particularly when global positioning system (GPS) measurements are not available. The dominant estimation tool remains the extended Kalman filter (EKF) applied to an inertial measurement unit (IMU). The growing availability of low-cost commercial-grade IMUs raises questions about how to best improve sensor readings into an estimate when measurements are available from multiple IMUs. This paper evaluates four different approaches to attitude estimation from multiple IMU measurements and their application in high dynamic motion. The four approaches are fusion of measurements (virtual IMU), fusion of state estimates (Federated KF), feedback fusion state estimate (Feedback Federated KF), and an EKF design incorporating the additional measurements (Augmented KF); these correspond to fusion before, within, or after state estimation. The performance of the approaches is quantified for varying IMU number theoretically and experimentally. The experiments use onboard autopilot hardware implementations of the estimators during motion in a motion capture volume and the peak and root-mean-square (RMS) errors used to quantify accuracy. The RMS error results indicate that the feedback federated Kalman Filter using five IMUs returns 38% compared to general federated Kalman Filter using 37% accuracy improvement over a single IMU. This improvement compares to a 19% improvement for virtual IMU and 9% improvement for the Augmented KF respectively. These results indicate that the Federated KF approach achieves the lowest RMS error relative to the virtual IMU and augmented KF approaches and inform the design of multi-IMU UAS pose estimators. The estimators are than used as a part of wind estimation without airflow sensor using two different methods, one using direct method (GPS+INS), and IMU disturbance method.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

In recent years, the growth in commercial applications of Unmanned Aerial Systems (UAS) have been supported by the availability of inertial measurement units (IMUs). IMU manufacturing processing improvements have resulted in reductions of size, price and power consumption, combined with software and algorithm development including sensor calibration, measurement integration, sensor fusion. These advances have all supported the proliferation of consumer grade IMUs in low-cost UAS platforms.

As commercial UAS applications extend to becoming measurement system platforms, there is a need for improved state estimation accuracy without significant cost increase. A key use of the estimates are as a reference for atmospheric wind inference algorithms that use precise state information to estimate local wind conditions [21]. When the wind estimates are used as inputs to developing local weather forecasting algorithms that have sensitive dependence on on initial conditions, the precision of state estimation must be improved beyond the tolerances acceptable for UAS navigation. Given the decreasing cost of consumer and industrial grade IMUs, multi-IMU estimators geared towards improving accuracy can yield a significant improvement in accuracy at a low cost and serve as a foundation to long term dead reckoning, real-time accurate wind and environment parameter estimations, and potentially provide tactical grade applications using consumer grade sensors.

There are multiple options to fusing sensor information, and this study considers the fusion problem by considering four basic approaches: fusing the measurements (i.e., before estimation), higher order estimator design (i.e., within estimation), fusing the state estimates (i.e., after estimation), and fusing the state estimates with feedback. The main objectives of this thesis are to:

- Outline the four classes of multi-IMU frameworks for improving state estimation accuracy.
- Define underlying local attitude filters to be integrated with the different multi-IMU estimation approaches.
- Test the performance of the system considered in software and analyze the results as a comparison of performance improvements against a reference.
- Use the multi-IMU estimation to study the improvement in wind field estimation and utilize it to test it in tornadoic simulated environment.

In this study, four general multi IMU state estimation frameworks are considered to fuse state estimates using two separate local Kalman filters. We than analyze and quantify the performance of the sensor fusion by implementing the multi-IMU state estimation using a hardware example with five IMUs.

1.1 Previous work

The idea of using multiple IMUs to achieve performance improvement has been used in different industries. [28] used an ad-hoc Kalman filter integrating two IMUs for marine navigational purposes. Position estimation accuracy from multi-IMU approaches can significantly exceed the single IMU case, as indicated by [8], which compared satellite launcher position estimation and showed the three IMU case reduced error by 54% relative to the single IMU case. [24] applied the [8] framework to an integrated navigation system that includes a traditional inertial navigation system (INS) with auxiliary IMU sensors.

Multiple researches such as the ones discussed in the papers [41], [42], [47], [52], [19], [16] have been conducted to improve noise characteristics using multi-IMU sensor arrays rather than focusing on sensor fusion for estimation improvement. One of such method is considered in this research by fusing the measurements into a synthetic IMU output or "virtual IMU" having reduced noise statistics [49]. This method is non domain specific as categorised in [15]. This means that this methods can be used for any application regardless of system.

Previous work in the aviation industry using multiple IMUs in a navigation system has been dominated by a focus on their applications as a part of estimator health monitoring or fault detection. In these treatments, their purpose is to facilitate sensor (IMU) or estimator fault detection rather than improve estimate accuracy. Examples of this work is outlined in [36]. Marine applications have also used multi-IMU (2-IMUs) for fault detection and redundancy in case of a single IMU failure such as in [39].

Some results are available for position-only multi-IMU pedestrian navigation architectures when a global positioning system (GPS) measurement is also available. [6], and [42] evaluated multiple and experimentally evaluated the architectures using five IMUs, which showed position estimate accuracy improvements exceed 30%.

Other studies that are more closely related to this research have focused on dynamic systems analyzed all frameworks individually rather than an extensive framework comparison [22], [46]. Where as other researches focused on the orientation of sensor placement [10], [20], [9], [34]. Despite the work in this area to develop multi-IMU fusion approaches, experimental data for dynamic systems on their comparative performance remains sparse. As a result, the question of which approaches are best implement in an airborne UAS platform requiring higher accuracy is not yet answered quantitatively for high rate dynamic systems. This work begins to answer that by systematically outlining and comparing four estimation strategies based on previous literature and most commonly referenced [5, 6, 8, 49]. The results include both theoretical and experimental results of the approaches implemented in an on-board UAS autopilot and on the same measurements, allowing direct comparison.

1.2 Thesis structure

The remainder of this thesis is structured as follows. Section 1.1 reviews previous work done in multi-IMU based navigation and their applications to different fields not limited to aviation. Section 2.1 defines the typical non linear system to be estimated using multi-IMU formulations. In Section 2.2, four different fusion approaches are considered and evaluated. Section 2.3 expands on the local filter design needed to implement the multi-IMU formulation. Section 4.2 outlines the experimental setup and implementation to evaluate and compare the performances of all the multi-IMU formulations and discusses the outcome of the study.

CHAPTER II

Estimator fusion formulation and error analysis

2.1 Problem statement

A general non linear system may be written in discrete dynamics form as

$$
\begin{aligned}
x_{k+1} &= f(x_k, u_k) + w_k \\
y_{i,k} &= h(x_k, u_k) + v_{i,k},
\end{aligned}
\tag{2.1.1}
$$

where the initial state x_0 is unknown Gaussian variable; i.e., $x_0 \sim \mathcal{N}(\mu_0, P_0)$. The state vector $x_k \in \mathbb{R}^n$, and control vector $u_k \in \mathbb{R}^m$. The measurement vector $y_{i,k} \in \mathbb{R}^p$ with i being the i^{th} sensor where $i = 1, 2, 3, \ldots, N$. The nonlinear dynamic and measurement function are represented by $f(\cdot) \in \mathbb{R}^n$ and $h(\cdot) \in \mathbb{R}^p$ respectively. The system process and measurement noise vectors $w_k \sim \mathcal{N}(0, Q)$ and $\nu_{i,k} \sim \mathcal{N}(0, R)$ assume zero mean Gaussian noise with covariance matrices $Q \in \mathbb{R}^{n \times n}$ and $R_i \in \mathbb{R}^{p \times p}$.

The contents of this thesis are submitted to Proceedings of the 2021 IEEE Access Journal [33]

$$E\left(\begin{bmatrix} w \\ v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}\begin{bmatrix} w \\ v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}^T\right) = \begin{bmatrix} Q & 0 & 0 & 0 & 0 \\ 0 & R_1 & 0 & 0 & 0 \\ 0 & 0 & R_2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & R_N \end{bmatrix} \tag{2.1.2}$$

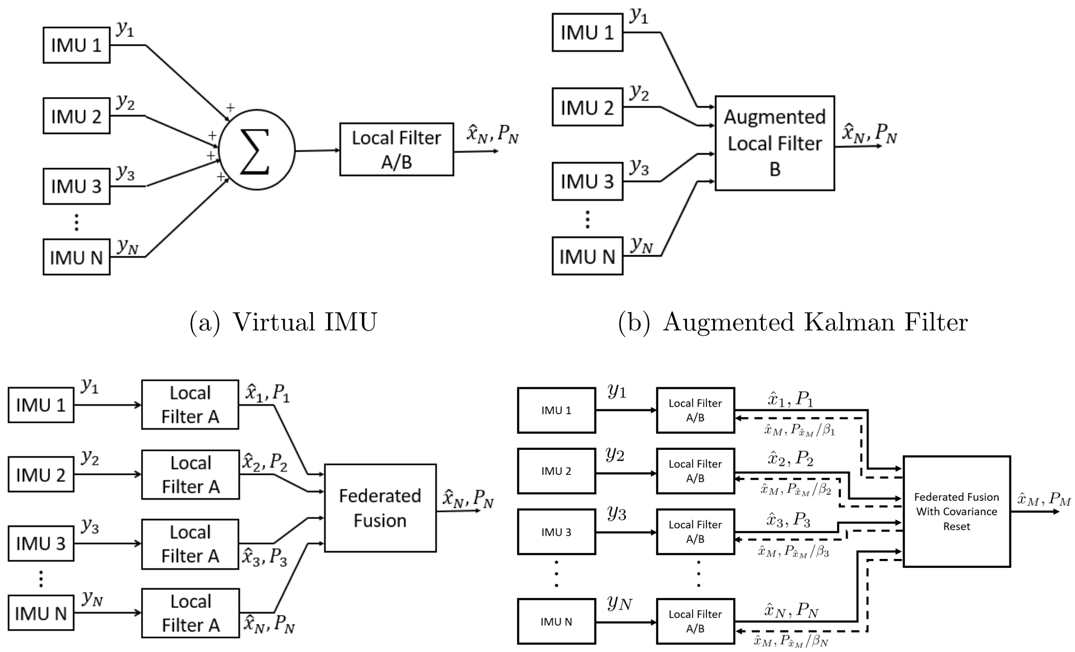
For the given problem our aim is to find the optimal (in a minimum variance sense) fusion technique to obtain $\hat{x}_m(k)$ based on measurements $y_i(k)$, where $i = 1, 2, \ldots, N$ which satisfies following minimum performance requirements:

1. The fused state estimate is unbiased.
2. Optimal weights for sensor fusion minimize the trace of error covariance.

This study explores four approaches to this problem, which are referred to as the federated Kalman filter (FKF), The augmented Kalman filter (AKF), and the virtual inertial measurement unit (vIMU) approach.

2.2 Estimator fusion formulation and error analysis

In this section, the four different estimation fusion formulation are evaluated for their performance in multi IMU framework as shown in Fig. 1. The virtual IMU method is most consistent with previous single-sensor estimates and this configuration used to define a theoretical "ideal" improvement benchmark which is used to evaluate the performance of actual implementation. All four frameworks described are globally optimal or sub-optimal depending on embodiment constraints [13], [12], [35].



(a) Virtual IMU

(b) Augmented Kalman Filter

(c) Federated Kalman Filter

(d) Feedback Federated Kalman Filter

Figure 1: Estimation fusion architectures considered

2.2.1 Virtual Inertial Measurement Unit (vIMU)

The virtual IMU approach first unifies the n sensor measurements into a single measurement having improved noise characteristics [45], [6] by an arithmetic mean of measurements from each sensor. The approach then applies a traditional single IMU sensor estimator and uses its output as the fused state estimates, as shown in Fig. 1(a). More precisely, the estimator is constructed as

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + \Gamma_k w_k \\ y_k &= \frac{1}{N}\sum_{i=1}^{N} y_{i,k} \end{aligned} \tag{2.2.1}$$

where x_k is a n dimensional state, with Φ_k and Γ_k being discrete state and noise transition matrix respectively. $y_{i,k}$ denotes the measurements obtained from each sensor with N being total number of sensors.

An idealized estimate of the improvement in state estimation for the VIMU approach may be derived by applying Bernoulli's theorem, or the weak law of large numbers [31], which describes how a sequence of probabilities converges. Under the assumption of multiple measurements being independent variables drawn from the same distribution, the law describes the behavior of the average of the results obtained from a large number of trials. The mean result approaches the distribution's expected value and application of the Chebyshev inequality [27] shows the result will tend to become closer to the expected value as more trials are performed. Assuming the measurements $\hat{x}_1$,$\hat{x}_2$,...,$\hat{x}_N$ are N independent state estimates of equal variance ($\sigma^2_{\hat{x}_N}$), the sample mean $\hat{x}$ approaches the true state as $N \longrightarrow \infty$.

$$
\begin{aligned}
\operatorname{var}(\hat{x}_m) &= \operatorname{var}\left(\frac{\hat{x}_1 + \hat{x}_2 + \cdots + \hat{x}_N}{N}\right) \\
&= \operatorname{var}\left(\frac{\hat{x}_1}{N}\right) + \operatorname{var}\left(\frac{\hat{x}_2}{N}\right) + \cdots + \operatorname{var}\left(\frac{\hat{x}_N}{N}\right) \\
&= \frac{\sigma^2_{\hat{x}_1}}{N^2} + \frac{\sigma^2_{\hat{x}_2}}{N^2} + \cdots + \frac{\sigma^2_{\hat{x}_N}}{N^2} \\
&= \frac{\sigma^2_{\hat{x}_N}}{N}
\end{aligned}
$$

Given, for any $\epsilon > 0$

$$
\lim_{N \to \infty} P\left[\left|\frac{\hat{x}_m}{N}| - \hat{x}_N\right| \geq \epsilon\right] \to 0
$$

From Chebyshev's inequality,

$$
P[|\hat{x}_m - \hat{x}| \geq \epsilon] \leq \frac{\operatorname{var}(\hat{x}_m)}{\epsilon^2} = \frac{\sigma^2_{\hat{x}_N}}{N\epsilon^2}.
$$

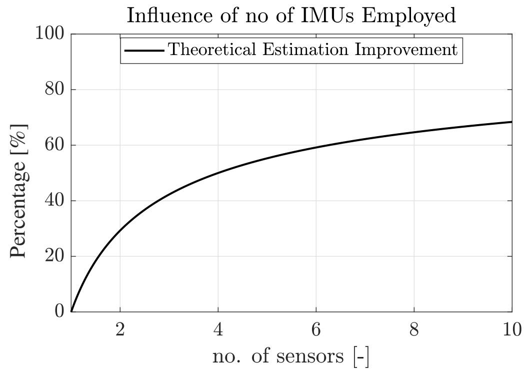


Figure 2: Percentage improvement in estimation with numbers of IMUs employed.

Thus,

$$\lim_{N\to\infty} P[|\hat{x}_m - \hat{x}_N| \geq \epsilon] = 0,$$

giving the ideal improvement in state estimation one can expect to be

$$\begin{aligned} \sigma_{\hat{x}_m} &= \sqrt{\mathrm{var}(\hat{x})}, \quad \text{or} \\ \sigma_{\hat{x}_m} &= \frac{\sigma_{\hat{x}_N}}{\sqrt{N}}. \end{aligned}$$

This idealized estimate accuracy improvement, shown in Fig. 2, serves as a theoretical contour with which to compare the measured performance improvement of the multi IMU estimators.

2.2.2 Augmented Kalman filter

The augmented Kalman filter approach consists of designing an extended Kalman filter for the problem using the augmented measurement vector

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

consisting of all N available measurements as shown in Fig. 1(b). The corresponding discrete measurement equation may be written as

$$y_k = H_k \hat{x}_k \tag{2.2.2}$$

where,

$$H_k = \begin{bmatrix} H_{1,k} \\ H_{2,k} \\ \vdots \\ H_{N,k} \end{bmatrix} \tag{2.2.3}$$

with H_1, H_2,. . .,H_n corresponds to their respective measurement matrix for each sensor (IMUs for scope of this research).

The nine parameter augmented estimation consists of a traditional propagation and measurement correction step [35]. The Augmented Kalman Filter differs from the typical single-IMU Kalman filter only in the measurement equation and measurement correction step, where it uses all gyro and accelerometer measurements. For the five IMU case tested in Section 4.2, the system has 30 observations and operates at the same frequency as the incoming observations. The observability matrix in equation (2.2.3) is full rank and the AKF is a semi-optimal estimator [35].

2.2.3 Federated Kalman Filter (FKF)

In the federated Kalman Filter approach shown in Fig. 1(c), N individual local state estimators are implemented, each having a single sensor (IMU) as an input and each generating both a state estimate $\hat{x}_i$ and corresponding covariance matrix $P_i, i = 1, ..., N$. As defined by Carlson [14], the approach

1. Scales the initial values of local filter covariance and process noise matrices.
2. Performs local time propagation and measurement update process.
3. Combines the updated local information into a global information.
4. Resets local information to the scaled global information.

The state estimates are then fused into a single state estimate $\hat{x}_m$ using a covariance-based weighting as

$$\hat{x}_m = P_m \left(\sum_{i=1}^{N} P_i^{-1} \hat{x}_i \right), \tag{2.2.4}$$

$$P_m = \left(\sum_{i=1}^{N} P_i^{-1} \right)^{-1}. \tag{2.2.5}$$

The FKF approach yields a globally optimal estimate [13] and its error covariance follows the additive equation (Eqn. (2.2.5)).

2.2.4 Feedback Federated Kalman Filter (FFKF)

This approach is an extension of the Federated Kalman Filter with an added step of resetting the state and covariance to the fused parameters for all local filters with scaled multi-IMU covariance propagation using scaling factor β. Eqn. (2.2.6)-(2.2.9) represents the estimation routine. This estimation method should perform better than general FKF as it propagates higher accuracy fused state and scaled covariance.

$$\hat{x}_m = P_m \left(\sum_{i=1}^{N} \beta_i P_i^{-1} x_i \right), \tag{2.2.6}$$

$$P_m = \left(\sum_{i=1}^{N} \beta_i P_i^{-1} \right)^{-1}, \tag{2.2.7}$$

$$\sum_{i=1}^{N} \beta_i = 1, \tag{2.2.8}$$

$$Q_m^{-1} = \sum_{i=1}^{N} \beta_i Q_i^{-1}. \tag{2.2.9}$$

The drawback of this approach is that it could diverge if not properly tuned. One of the parameter that could result in divergence is the choice of β. This method of information sharing through a scaled feedback follows all the information conservation principle outlaid by Carlson et al. [14] and followed by Brown et al. [12] shows that this method of state estimation follows the same optimality as the general federated Kalman filter.

The choice of β depends on a lot of factors such as . For this study we use the variance measurement based on (Appendix B) and [2] to conclude that the sensors used in this study are similar in performance and this approach is more direct than using condition number of covariance as in many studies and reduces computational complexity and hence, $\beta = 1/N$ is a reasonable assumption to make.

2.3 Local Filter Formulation

Due to difference in implementation requirements for all frameworks described in this study (i.e. AKF formulation measurements from sensors cannot be used during priori step), we need to use two different local filter methods. We use Bortz equation (2.3.2) to obtain an Extended Kalman Filter framework for attitude estimation to implement Virtual Sensor Method and Federated Kalman Filter, and we will use six degree of freedom (6DOF) kinematic equation to obtain a Extended Kalman Filter to be use to implement Augmented Kalman Filter. Both local filter formulation are derived from base sensor dynamics so the difference in performance just due to local filter is not observed.

2.3.1 Attitude Estimation using Bortz Equation.

The local attitude estimator algorithm used in this study is based on the symmetry-exploiting method proposed in Bortz [11]. For any attitude described by a quaternion $\mathbf{q}$, there exists a rotation vector ϕ such that

$$\mathbf{q}(\phi) = \begin{bmatrix} \frac{1}{2}\left(\frac{\sin(\gamma/2)}{\gamma/2}\right)\phi \\ \cos(\gamma/2) \end{bmatrix} \in \mathbb{R}^{4x1}. \tag{2.3.1}$$

The Bortz equation for rotation error as a function of angular rate ω may be written as

$$\dot{\phi} = \omega + \frac{1}{2}\phi \times \omega + \frac{1}{\gamma^2}\left[1 - \frac{\gamma \sin\gamma}{2(1-\cos\gamma)}\right]\phi \times (\phi \times \omega), \tag{2.3.2}$$

where ϕ is the rotation vector in Eqn. (2.3.1) and $\gamma = |\phi|$. Assuming small γ and neglecting higher order terms, Eqn. 2.3.2 becomes

$$\dot{\phi} = \omega + \frac{1}{2}\phi \times \omega. \tag{2.3.3}$$

Equation 2.3.3 now can be augmented with constant bias states $\boldsymbol{b}$ to form the state equation

$$\begin{bmatrix} \dot{\phi} \\ \dot{\boldsymbol{b}} \end{bmatrix} = \begin{bmatrix} \omega + \frac{1}{2}\phi \times \omega \\ \mathbf{0}_{3\times1} \end{bmatrix}. \tag{2.3.4}$$

Both EKF framework were derived by Pittelkau [35] for a recursive implementation and using the same notation for multiplicative product operator ($\otimes$), where *priori* state estimate in quaternion form can be estimates as per

$$\begin{bmatrix} \delta\hat{\phi}_k \\ \delta\hat{\mathbf{b}}_k \end{bmatrix} = \begin{bmatrix} \delta\hat{\phi}_{k-1} \\ \delta\hat{\mathbf{b}}_{k-1} \end{bmatrix}$$

with

$$\begin{bmatrix} \delta\hat{\phi}_{k-1} \\ \delta\hat{\mathbf{b}}_{k-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and the estimate quaternion $\hat{q}$ for that state is obtained as

$$\hat{q}_k^- = q(\hat{\phi}_k^-) \otimes \hat{q}_{k-1}^+.$$

Now, considering a vector $\nu^s = [\nu_x^s, \nu_y^s, \nu_z^s]^T$ in the sensor reference measurement. The measurement function for a three-axis magnetometer can be formulated as

$$y = \nu^s + \epsilon, \tag{2.3.5}$$

with ϵ as additive noise. The three-axis magnetometer is sufficient to generate an orientation estimate based on the magnetic field, which will be used in the correction step. All three axes are measured, thus $\partial \boldsymbol{h}/\partial(\nu^s)^T = \mathbf{I}$. The 3D measurement sensitivity matrix $H^{(3)} \in \mathbb{R}^{3\times 3}$ is given by

$$H^{(3)} = \left.\frac{\partial \nu^s}{\partial \phi^T}\right|_{\phi=0} = T_b^s \nu^b$$

where T_b^s is a body-to-sensor transformation matrix and ν^b is measurement in body frame.

The predicted measurement is than given by equation 2.3.5

$$\hat{y}_k = \hat{\nu}_b^s = q(\hat{\phi}_s^b)\nu^s$$

where $q(\hat{\phi}_s^b)$ is the *priori* estimated attitude and the residual is simply $\nu_k = y_k - \hat{y}_k$. Using this information we can use the sub-optimal Kalman Gain to obtain the *posteriori* state estimates using measurements obtained from magnetometer readings as a standalone correction to the EKF output from the IMU estimates.

$$K_k = P_{k-1}H_k^T(H_k P_{k-1} H_k^T + R_k)^{-1}$$

$$\hat{\phi}_k^+ = \hat{\phi}_k^- - K_k(y_k - \hat{y}_k),$$

where, K is the Kalman gain and P is the covariance matrix. Now, with updated state estimate in quaternion form is given by

$$\hat{q}_k^+ = q(\hat{\phi}_k^+) \otimes \hat{q}_{k-1}^-.$$

Similarly, the formulation outlined by Koshravian [25] can be used to integrate information about state estimates using heterogeneous sensors like LIDAR, optic-flow, etc.

2.3.2 Attitude Estimation using Kinematic Equation.

Using formulation for attitude estimation by Kane and Levinsion [23] using quaternions we can represent the non linear system observer in equation (2.1.1) using gyroscope angular measurement to describe the quaternion dynamics and bias as random walk as:

$$\begin{bmatrix} \dot{q} \\ \dot{b}^{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} S(\omega - b^{\omega}) q \\ 0 + \nu \end{bmatrix} = \begin{bmatrix} \frac{1}{2} S(q)(\omega - b^{\omega}) \\ 0 + \nu \end{bmatrix} \tag{2.3.6}$$

and

$$S(\omega) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

$$S(q) = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix},$$

where ν is noise, $q = [q_0, q_1, q_2, q_3]$ are quaternion representing the orientation of the vehicle, and $\omega = [\omega_x, \omega_y, \omega_z]$ are bias corrected gyro measurements. The accelerometer and magnetometer are then used as measurement to compensate for drift from gyro bias error as:

$$\begin{bmatrix} y_h^a \\ y_h^m \end{bmatrix} = \begin{bmatrix} C_L^I(a_e - g) + e^a \\ C_L^I B^N + e^m \end{bmatrix}, \tag{2.3.7}$$

with C_L^I defines the rotation from body L to intermediate reference frame I.

Taking the nonlinear system's Jacobian to linearize and discretize with time step of $dt = t_k - t_{k-1}$ gives

$$\underbrace{\begin{bmatrix} q \\ b^\omega \end{bmatrix}_k}_{x_k} = \underbrace{\begin{bmatrix} I_{4\times4} & -\frac{dt}{2}S(q) \\ 0_{3\times4} & I_{3\times3} \end{bmatrix}_{k-1}}_{\Phi_{k-1}} \underbrace{\begin{bmatrix} q \\ b^\omega \end{bmatrix}_{k-1}}_{x_{k-1}} \tag{2.3.8}$$

and

$$\underbrace{\begin{bmatrix} y_h^a \\ y_h^m \end{bmatrix}_k}_{y_k} = \underbrace{\begin{bmatrix} C_a & 0_{3\times3} \\ C_m & 0_{3\times3} \end{bmatrix}_k}_{H_k} \underbrace{\begin{bmatrix} q \\ b^\omega \end{bmatrix}_k}_{x_k} \tag{2.3.9}$$

with,

$$C_a = -2\begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & q_0 & q_3 & q_2 \\ q_0 & -q_1 & -q_2 & q_3 \end{bmatrix}_k$$

$$C_m = -2\begin{bmatrix} q_3 & q_2 & q_1 & q_0 \\ q_0 & -q_1 & q_2 & -q_3 \\ -q_1 & -q_0 & q_3 & q_2 \end{bmatrix}_k$$

Now the optimal estimate for the state vector can be obtained using Kalman filter using time update and observation update. The time update process of the Kalman filter is independent and is written as outlined by Yang and Gao [50]:

$$\begin{aligned}\hat{x}_{k|k-1} &= \Phi_{k-1}\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= \Phi_{k-1}P_{k-1|k-1}\Phi_{k-1}^{T} + Q_k.\end{aligned} \tag{2.3.10}$$

The observation update equation of the Kalman filter is expressed as:

$$\begin{aligned}K_{k|k} &= \frac{P_{k|k-1}H_k^T}{(H_k P_{k|k-1} H_k^T + R_k)} \\ P_{k|k} &= (I - K_{k|k}H_k)P_{k|k-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_{k|k}(y_k - H_k\hat{x}_{k|k-1})\end{aligned} \tag{2.3.11}$$

where $\hat{x}_{k|k-1}$ is the a priori state estimation, $\hat{x}_{k|k}$ is the a posteriori state estimation, $K_{k|k}$ is the Kalman gain matrix of the Kalman filter, $P_{k|k-1}$ is the a priori covariance matrix of the state vector, $P_{k|k}$ is the a posteriori covariance matrix of the state vector, R_k is the covariance matrix of the measurement noise vector, Q_k is the covariance matrix of the process noise and Φ_k is the system transition matrix from time $k-1$ to time k.

CHAPTER III

Wind estimation

Every year, severe weather threatens the lives of many across the United States and the world. A lot of research has been conducted in recent years to predicting and improve warning time for residents about potential severe thunderstorms. This surge in research is because of high false alarm rate (FAR) that is at 48% with a probability of detection (POD) of 81% as of 2013. However, when these predictions extend to include tornado statistics, the FAR increases to 74% and the POD decreases to 57% [37]. The severe storm prediction has leveled out with current prediction methods. The tornado predictions, however, are on a downward trend despite the use of modern meteorological equipment and techniques (Doppler radar, supercomputer-aided weather model forecasting, etc.). The inability to reliably predict tornado genesis is largely due to the lack of vital data that predicates the formation and path of a tornado.

There has been a lot of research in vehicles that could withstand the harsh condition of a thunderstorm and even tornado. Development of such vehicles ([37], [4]) opens doors for remote and on board data collections of information that could lead to very useful outcomes in weather prediction as specially as events such as tornado which require a lot of local data to reliably predict and track them.

There has been some research for UAS performance in server storm weather such as done by

Eric Flew et al [18], but most of the studies like this focus on the large time scale phenomenon and sensing real-time deviations, and onboard monitoring of aircraft energy state and health. Eric Flew et al [18] describes a networking approach to maintain a robust communication architecture that can provide situational awareness to UAS, provide real-time telemetry and control for operators, the return of sensor data, and establish, monitor, and maintain, UAS in high dynamic situations but fails to address reliability of wind speed estimation obtained.

Due to difficulty of obtaining true tornadic wind speeds it is very hard to verify the working of most commonly used algorithms in such conditions. This study introduces an approach for wind field estimate validation in a simulated tornado, defines a procedure of model comparison and validation of tornado using hardware-in-loop simulation.

This study's approach to evaluating wind field estimation techniques relies on sensor data gathered from simulated flights through a representative wind field. This coupled environmental and flight dynamics simulation was accomplished through (a) defining two idealized wind fields (tornado with and without gusts), (b) implementing a simulation routine that incorporates both the wind field and sensor noise models, and (c) simulating arbitrary trajectories through the wind field while a wind estimator technique observes the sensor outputs. The sensors modeled included GPS (position and velocity) and IMU (accelerometer and rate gyro) measurements with experimentally determined noise characteristics. The wind estimates are compared to the true wind speeds to quantify wind estimation error.

3.1 Tornado modeling

The tornado wind field model used for this study is the one proposed by Ash et. al [3], for which theoretical acoustic solutions have been developed. Because acoustic emissions represent a very small portion of the energy in a wind field, the ability to measure acoustic emissions of a fluid model provides an additional mechanism to verify its similitude, and contemporary work in airborne acoustic measurement of tornadoes will benefit from the availability of a flight simulation with an implemented tornado model that has acoustic relevance.

The model relies on an eddy viscosity turbulence model which only influences the steady-state vortex velocity field in the vicinity of the core. Moreover, it uses the results outline by Saffman [38] that the Burgers velocity is one class of solenoidal vorticity and the solution is governed by

$$\frac{D\omega}{Dt} = (\omega.\nabla)U + \nu\nabla^2\omega \tag{3.1.1}$$

$$U_i = \alpha_{ij}x_j, \text{with } \alpha_{ii} = 0, \tag{3.1.2}$$

where U_i denotes the mean velocity components. In this equation, the vortex stretching mean velocity stabilizes the Burgers vortex, and avoids the transient Lamb-Oseen vortex behaviour.

Taking a finite diameter Scorer [40] potential vortex that propels the flow giving an solenoidal vorticity, and utilizing the AZZ core radius and maximum swirl velocity to define reference circulation $\Gamma_\infty = 4\pi R_{core}V_{\theta,max}$ azimuthal velocity distribution is represented as,

$$V_{\theta,AZZ(r)} = 2V_{\theta,max}\frac{\dfrac{r}{R_{core}}}{1+\left(\dfrac{r}{R_{core}}\right)^2} \tag{3.1.3}$$

Figure 3 illustrates the non equilibrium AZZ vortex defined above for R_{core} = 100m and $V_{\theta,max}$ = 50m/s.

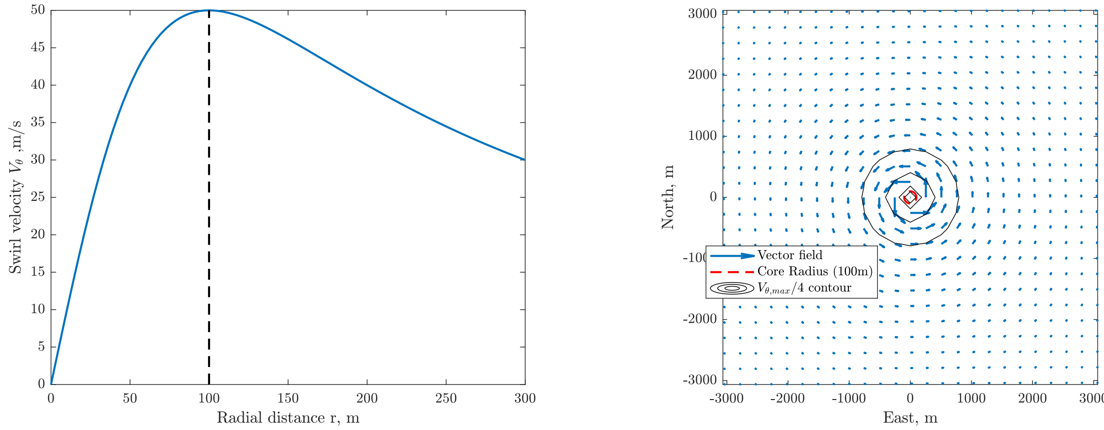


Figure 3: Azimuthal velocity profile (left), and top view (right) of wind field generated by non-equilibrium AZZ vortex

3.2 Spatio-temporal interpolation

In a deployed onboard wind field estimator measuring winds through a tornado, the onboard tornado wind field estimator must generate both local wind field estimates and regional estimates (eg, velocity distributions) using real-time onboard hardware. The onboard digital representation and solution requires the continuous system to be discretized, yet intermediate values must still be available for use in the onboard wind field estimates. As tornado modeling is a dynamic system, its variables change in both 3-dimensional space and in time as well. Thus, onboard deployment imposes different constraints than simulating an aircraft in tornadic wind field, and hence we require a way to obtain value of wind speeds at every time step of the aircraft's trajectory (e.g., arbitrary points in both space and time).

This problem can be solved using a 4D(3-space, 1-time) linear interpolation. One may derive the expressions for 4D interpolation is by assuming two malleable 3D shapes (box) which can translate in time and change in every dimension. This can be thought of as the first box to be at initial time and second the same box with changes in each dimension at second time step. Fig 4 is a simple visualization of the explanation.

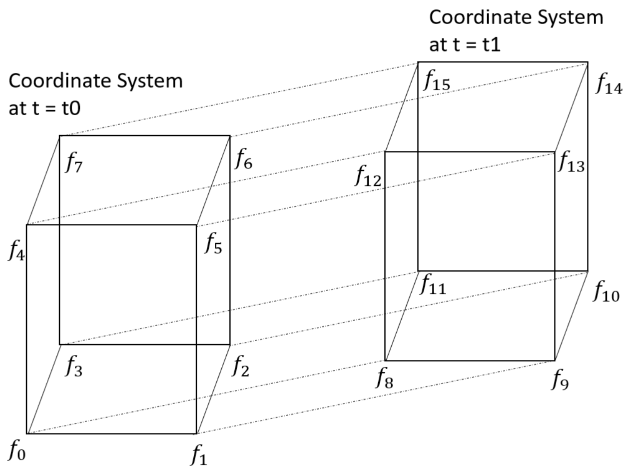


Figure 4: This visualization represents concept of 4D interpolation.

The equation for interpolation can be considered as an 4D Taylor expansion of first order, which is than evaluated for all 16 terms for each parameter u, v, w as shown.

$$f(t,x,y,z) = a_0 + a_1x + a_2y + a_3z + a_4t + a_5xy + a_6yz + a_7xz + a_8xt + a_9yt + a_{10}zt + a_{11}xyz + a_{12}xyt + a_{13}yzt + a_{14}xzt + a_{15}xyzt \tag{3.2.1}$$

where, f represents wind parameters at any given time and space coordinate. When evaluated at all 16 vertices of the box shown in Fig 4, we can set an linear matrix which results in the solution of all 16 coefficients which can than be used to find interpolated solution for any given coordinate.

$$
\begin{bmatrix} f_0(t_0, x_0, y_0, z_0) \\ f_1(t_0, x_1, y_0, z_0) \\ f_2(t_0, x_1, y_0, z_0) \\ \vdots \\ f_15(t_1, x_1, y_1, z_1) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & y_0 & \dots & x_0y_0z_0t_0 \\ 1 & x_1 & y_0 & \dots & x_1y_0z_0t_0 \\ 1 & x_0 & y_1 & \dots & x_0y_1z_0t_0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1 & y_1 & \dots & x_1y_1z_1t_1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_15 \end{bmatrix} \tag{3.2.2}
$$

simplifying, we get:

$$
A = F * X^{-1} \tag{3.2.3}
$$

where,

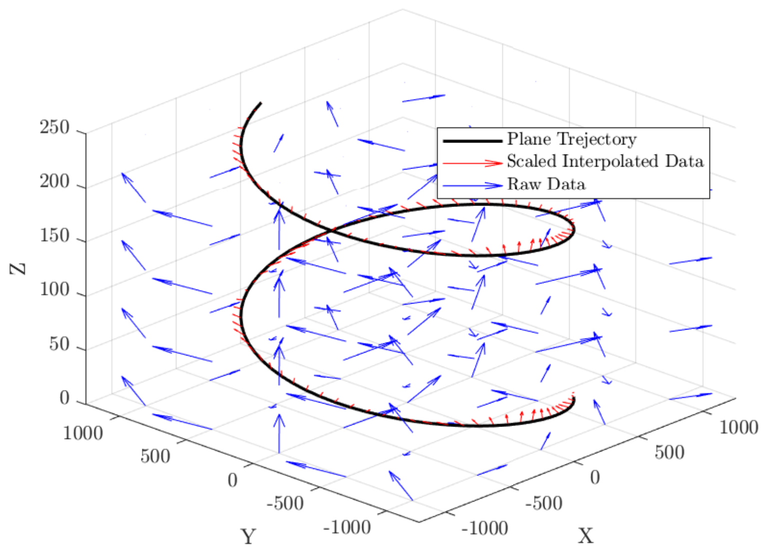
$$
A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{15} \end{bmatrix}, F = \begin{bmatrix} f_0(t_0, x_0, y_0, z_0) \\ f_1(t_0, x_1, y_0, z_0) \\ f_2(t_0, x_1, y_0, z_0) \\ \vdots \\ f_{15}(t_1, x_1, y_1, z_1) \end{bmatrix}, X = \begin{bmatrix} 1 & x_0 & y_0 & \dots & x_0y_0z_0t_0 \\ 1 & x_1 & y_0 & \dots & x_1y_0z_0t_0 \\ 1 & x_0 & y_1 & \dots & x_0y_1z_0t_0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1 & y_1 & \dots & x_1y_1z_1t_1 \end{bmatrix}.
$$

Fig 5 represents an example of an aircraft flying through a 4D varying wind field.

3.3 Wind estimation

3.3.1 Direct wind estimation method

Wind estimation method used in this study is considered a direct method where the GPS measurements and vehicle dynamics and kinematic equations are used to obtain orientation of the vehicle using kalman filtering which is discussed in the following section.



4

Figure 5: Example of an aircraft flying through a 4D varying wind field.

Equation of motion of the vehicle are briefly derived. Considering similar setup for vehicle frames as done by Langlaan [26]. Consider an aircraft located at $\mathbf{r}$ in an inertial frame $\mathbf{I}$, where $\hat{x}^i$, $\hat{y}^i$, and $\hat{z}^i$ define unit vectors as shown in Fig:6.

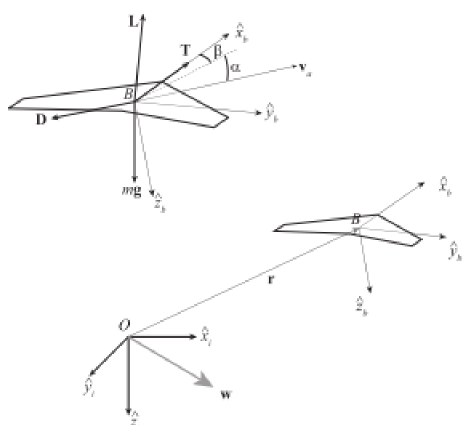


Figure 6: Reference frame setup as shown in Langelaan [26]

Based on fixed reference body frame, with body velocity v_a having component u, v, and w in body frame $\hat{x}_b$, $\hat{y}_b$, and $\hat{z}_b$ respectively. The velocity of the frame than can be written as

$$\dot{r} = \mathbf{v}_a + \mathbf{w}$$

Hence,

$$\ddot{r} = \frac{d}{dt}\mathbf{v}_a + \frac{d}{dt}\mathbf{w}$$

where,

$$\frac{d}{dt}\mathbf{v}_a = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \omega \times \mathbf{v}_a$$

substituting, $\omega = [p, q, r]^T$ and simplifying

$$\mathbf{X} + \mathbf{Y} + \mathbf{Z} + m\mathbf{g} = m[(\dot{u} + qw - rv)\hat{x}_b + (\dot{v} + ru - pw)\hat{y}_b + (\dot{w} + pv - qu)\hat{z}_b + \frac{d}{dt}w]$$

where $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, are aerodynamic forces in the body x, y, z directions, respectively and m$\mathbf{g}$ is the force due to gravity. The vector of wind accelerations $\frac{d}{dt}w$ is expressed in the inertial frame. Using a direction cosine matrix $\mathbf{T}$ which transforms a vector expressed in the inertial frame to a vector expressed in the body frame and simplifying using linearization of equation we end up with:

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

and,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \mathbf{T}^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} w_{i,x} \\ w_{i,y} \\ w_{i,z} \end{bmatrix} \tag{3.3.1}$$

Now, GPS provides a direct measurements of velocity in fixed reference frame. Thus the wind measurements can me made from the equation above.

$$\begin{bmatrix} w_{i,x} \\ w_{i,y} \\ w_{i,z} \end{bmatrix} s = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}_{GPS} - \mathbf{T}^{-1} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.3.2}$$

This is said to be a direct wind estimation method as the measurements are utilized directly. This method also makes an assumption that the attitude variables or Euler angles (ϕ,θ,ψ) are available independently regardless of wind estimation routine. For this study we will obtain the euler angles using kalman filter method as described in 2.3.

Wind Estimation using rate of change of wind velocity

The IMU provides measurement of beady-axis acceleration with respect to the inertial frame and the gravity vector projected into the body frame. This method of using IMU to get using rate of change of wind velocity or simply wind acceleration that is imprated on IMU was introduced by Jack Langeelan [17].

Using the wind acceleration formulation derived in the paper we get

$$\begin{bmatrix} \dot{w}_{x,b} \\ \dot{w}_{y,b} \\ \dot{w}_{z,b} \end{bmatrix}_{k-1} = \begin{bmatrix} a_{x,b} - b_{imu,x} - g\sin(\theta) \\ a_{y,b} - b_{imu,y} + g\sin(\phi)\cos(\theta) \\ a_{z,b} - b_{imu,z} + g\cos(\theta)\cos(\phi) \end{bmatrix}_{k-1} - \begin{bmatrix} qw - rv \\ pw - ru \\ qu - pv \end{bmatrix}_{k-1} - \frac{1}{2\Delta t}\begin{bmatrix} u_k - u_{k-2} \\ v_k - v_{k-2} \\ w_k - w_{k-2} \end{bmatrix} \tag{3.3.3}$$

Equations 3.3.3 and 3.3.2 together allows computation of the wind velocity (expressed in the inertial frame) and the wind acceleration as seen by the vehicle (expressed in the body frame). These can be used to compute velocity or trajectory commands for energy maximization or other tasks such as disturbance minimization. It is useful, however, to determine the expected error in wind field estimates.

CHAPTER IV

Simulation and experimental implementation

4.1 Estimation run

To evaluate the performance of all four estimation fusion routine, estimators had to track a manual oscillatory input in range of -90° to +90° (which includes gimbal lock singularity) at an angular rate of about 36 $°/sec$ in each axis. The data obtained from onboard estimation and motion capture system were time synced by an impulse strike at the start of test.

All the state estimators were initialized at zero state quaternion, with alignment correction using magnetometer. Conservative measurement noise covariances were chosen for the

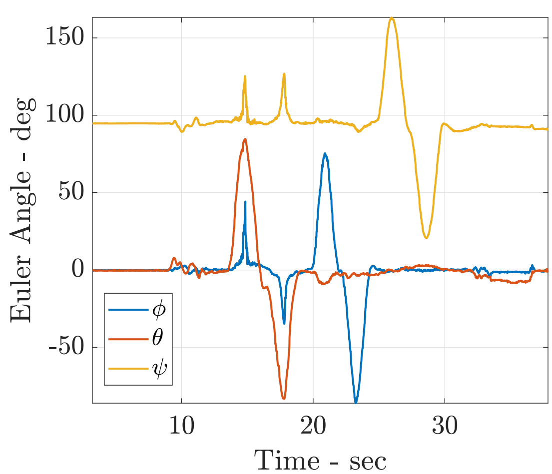


Figure 7: Truth data from motion tracker

Table 1: Motive OptiTracker System specification

Positional accuracy	± 0.2 mm
Rotational accuracy	± 0.1 deg

simulation.

For more intuitive interpretation and clear visual comparison all the plots described below are plotted in Tait-Bryan angles (body 3-2-1 SO(3) rotation [51])

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \mathtt{atan2}(2(q_2q_3 + q_1q_4), 1 - 2(q_3^2 + q_4^2) \\ -\mathtt{asin}(2(q_2q_4 - q_1q_3)) \\ \mathtt{atan2}(2(q_3q_4 + q_1q_2), 1 - 2(q_2^2 + q_3^2) \end{bmatrix}, \tag{4.1.1}$$

with ψ as the heading angle of the aircraft, θ as the pitch angle of the aircraft and ϕ as the roll/bank angle, and with the definition of quaternion is consistent with Eqn (2.3.1).

4.2 Experimental implementation

To illustrate the working and performances comparison of the outlined estimator fusions in this study, the estimation routines are implemented on an onboard unmanned aerial system (UAS) autopilot and compared to both onboard estimates using contemporary autopilots and external reference data. The UAS implements the proposed estimators using an onboard autopilot built on a Raspberry PI 3B+ platform running Navio2 which includes two IMUs (MPU9250 [48] and LSM9DS1 [44]). A Pixhawk 2.1 which includes additional three IMUs (2 MPU9250 and LSM303D [43]) running the contemporary Ardupilot estimator as well as logging raw IMU data is also mounted on the UAS and identifying markers are added to use an external motion capture system (OptiTracker) shown in Fig: (8) as a reference to expected results due to high accuracies as given in Table (1).

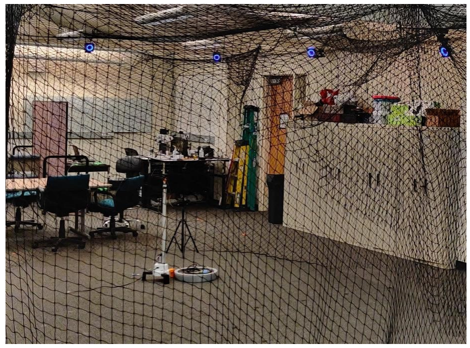


Figure 8: Experimental setup, including 24 camera motion capture system.

The data collected from all five IMUs at a rate of 200Hz are used as input to all the estimation fusion algorithms. To evaluate the performance of the estimation routines all the state estimates are compared to the output from the motion capture system measurement at the same rate of 200Hz.

4.2.1 Performance analysis in motion capture environment

To analyse the performance of each estimator fusion algorithms, described in this study we find Root Mean Square Error (RMSE) for each formulation and compare it against the data obtained from motion capture system using

$$RMSE = \sqrt{\sum_{i=0}^{i=N} \frac{(\hat{x}_i - x_i)^2}{N}}, \tag{4.2.1}$$

with $\hat{x}_i$ being the estimates obtained from the state estimator formulations and x_i are the corresponding states obtained using motion capture system.

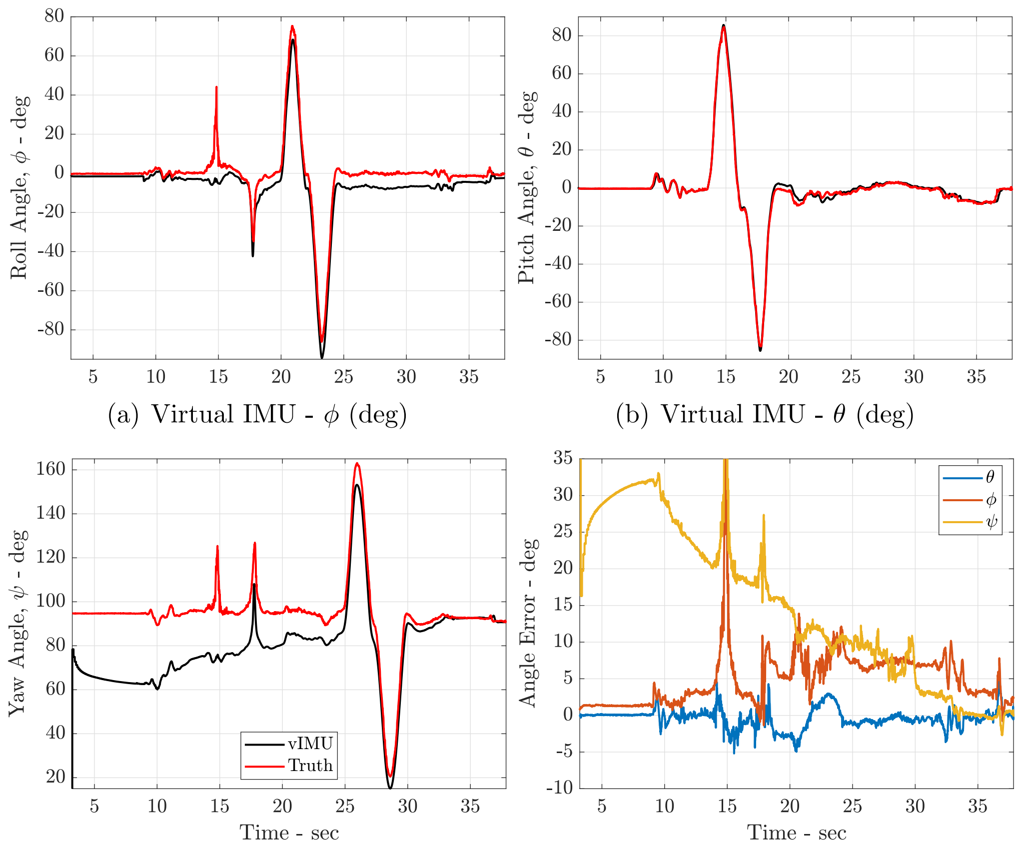
While motion capture was used as a reference, the formulation does not always represent the true attitude. In particular, the 43° pitch angle transient at 13sec does not reflect true attitude and is related to underlying coordinate frame definition differences between the estimation routines and the motion capture (Optitracker) system's Robot Operating System (ROS) toolkit (right and left handed coordinate systems). The ROS toolkit is in widespread contemporary use and the data has not been altered to provide a realistic comparison as can be odserved in Fig. (9)-(12).

The random walk and the rate random walk for the gyroscope were found using 12 hour data collection and analysis using basic allan variance as described in Appendix 5.1.1 of $\sigma_{IMU1} = 0.006°/\sqrt{s}$, $\sigma_{IMU2} = 0.016°/\sqrt{s}$, $\sigma_{IMU3} = 0.0206°/\sqrt{s}$, $\sigma_{IMU4} = 0.012°/\sqrt{s}$, and $\sigma_{IMU5} = 0.0231°/\sqrt{s}$.

Comparison of vIMU to motion tracking

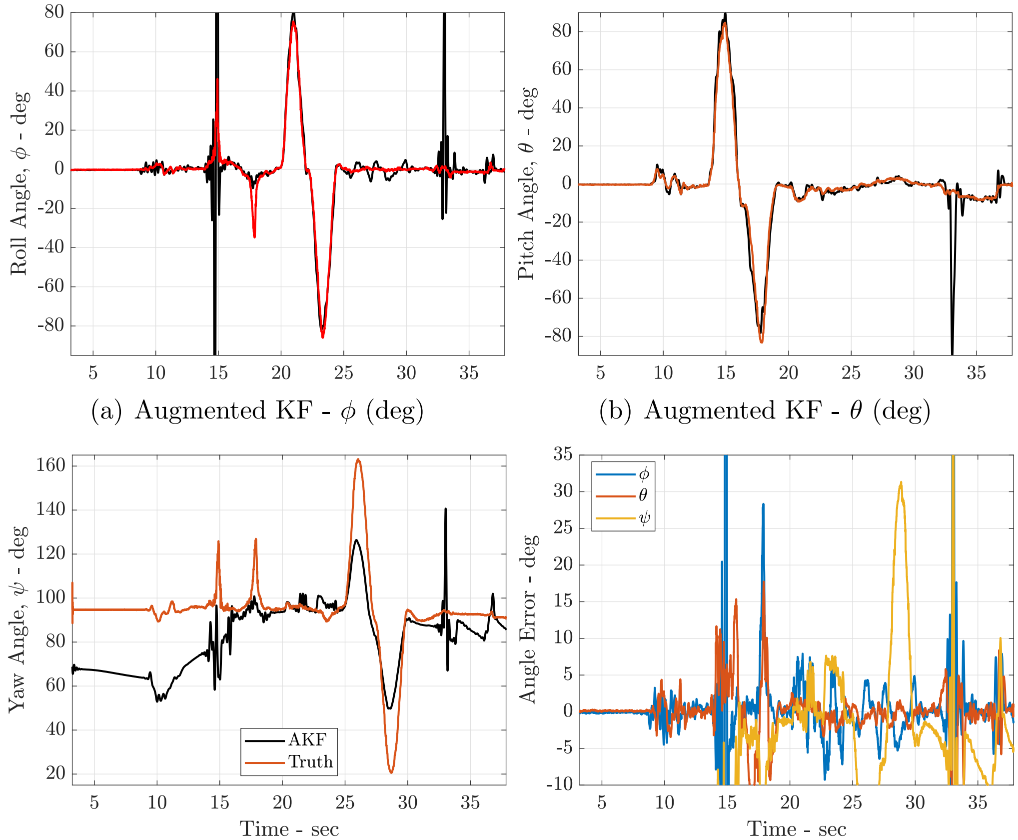
When running vIMU routine we combine all the measurements into one single measurement with expected noise reduction and hence, we only have one output for comparison. As there is only one local filter calculation taking place we expect computational load for this method to be minimum.

As shown in Fig. IV.9(a)- IV.9(c), the vIMU tracks θ very well compared to ϕ and ψ. The experiment was conducted indoors deviation from true ψ is expected. The RMSE values of all states using equation (4.2.1) are as shown in Fig. IV.9(d) and Table (4). We also see coupled effects between estimates. The deviation in ψ coupled with other estimates. This behaviour is expected as all the measurements were combined before the fusion and hence, there is only one input.



(a) Virtual IMU - ϕ (deg)
(b) Virtual IMU - θ (deg)
(c) Virtual IMU - ψ (deg)
(d) Virtual IMU - error (deg)

Figure 9: State comparison between Virtual IMU Filter and motion tracker



(a) Augmented KF - ϕ (deg)

(b) Augmented KF - θ (deg)

(c) Augmented KF - ψ (deg)

(d) Augmented KF - error (deg)

Figure 10: State comparison between Augmented Kalman Filter and motion tracker

Comparison of AKF to motion tracking

AKF is very similar in implementation as vIMU, but instead of combining measurements before fusing it to state estimates, AKF uses all the measurements simultaneously in its measurement matrix and is computationally heavier than vIMU. Even though it is computationally heavy the state estimation is not as accurate as seen from Fig. IV.10(a)- IV.10(c)

AKF deviates from its estimates during rapid motions near the time synchronization impulses. AKF estimates deviates significantly when ψ is far from the reference. The RMSE values of all states using equation (4.2.1) are as shown in Fig. IV.10(d) and Table (4).

Comparison of FKF to motion tracking

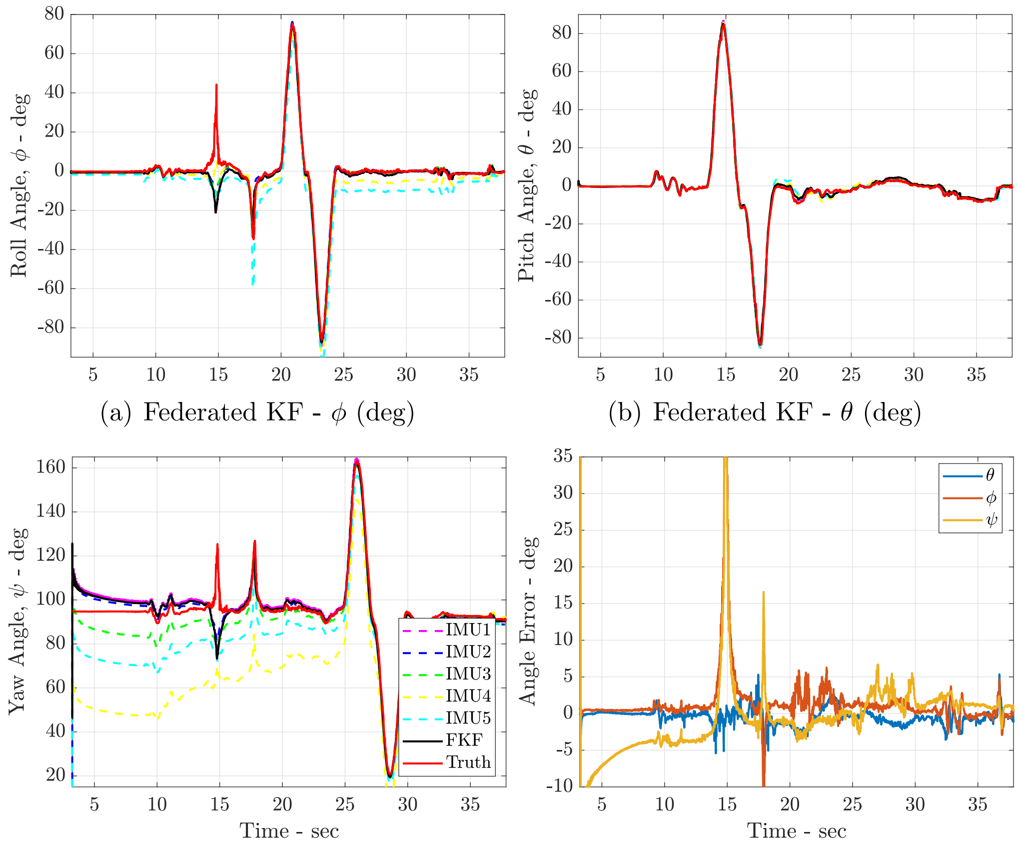
State estimates from the FKF are shown in Fig. IV.11(a)- IV.11(c). FKF best tracks the state estimates and even though experiment was conducted indoors FKF converges to the true ψ rapidly. The RMSE values of all states using equation (4.2.1) are as shown in Fig. IV.11(d) and Table (4). When compared to VIMU and AKF, FKF has lower estimate error. Moreover, as it uses the covariance matrix of each states to fuse the estimates there is no coupled deviations observed.

The FKF involves a larger number of computations than the other approaches as it requires individual state estimation and then is fused in one state estimation, but the advantage of FKF is that its structure allows easy integration of fault detection and individual sensor health monitoring.

Comparison of FFKF to motion tracking

The FFKF involves the same computational load as the FKF which is still larger than the other approaches as it requires individual state estimation and then is fused in one state estimation, FFKF holds all the advantages of FKF and still performs better than FKF.

State estimates from FKF are shown in Fig. IV.12(a)- IV.12(c). FFKF best tracks the state estimates and even though experiment was conducted indoors FFKF converges to the true ψ rapidly. The RMSE values of all states using equation (4.2.1) are as shown in Fig. IV.12(d) and Table (4). Since, FFKF is an extension of FFKF same behaviour is observed but with better states estimates.



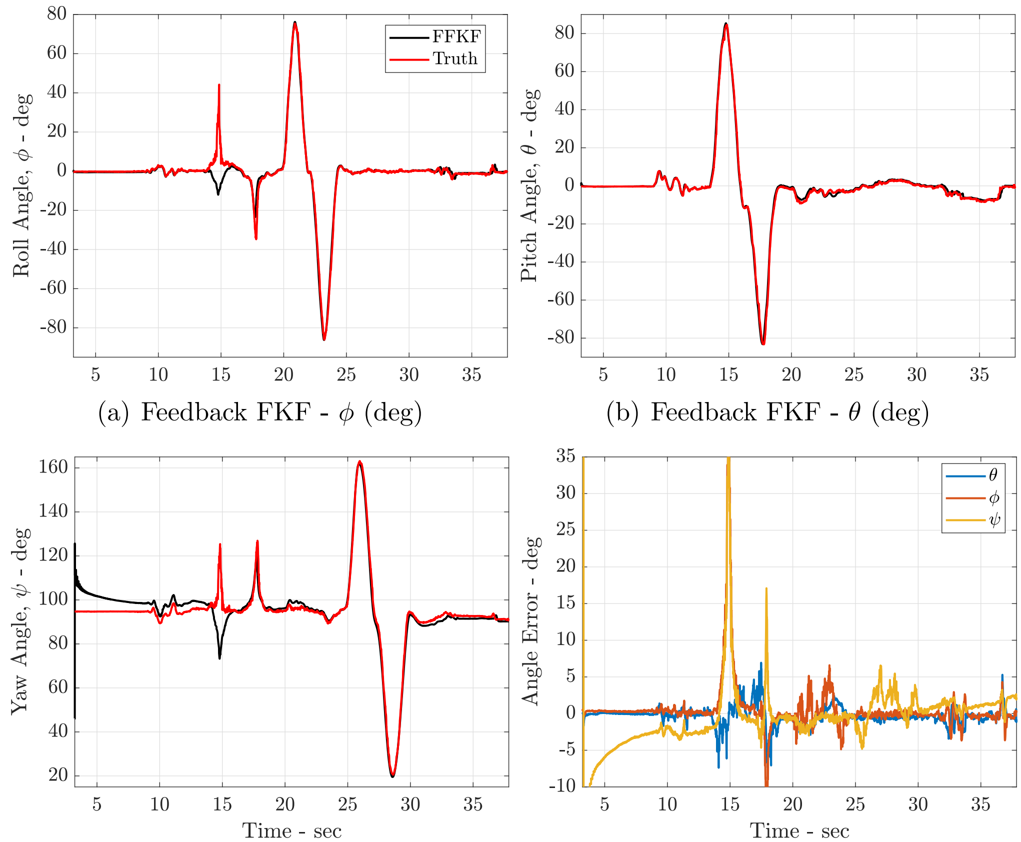
(a) Federated KF - ϕ (deg)

(b) Federated KF - θ (deg)

(c) Federated KF - ψ (deg)

(d) Federated KF - error (deg)

Figure 11: State comparison between Federated Kalman Filter, single IMU run, and motion tracker



(a) Feedback FKF - ϕ (deg)

(b) Feedback FKF - θ (deg)

(c) Feedback FKF - ψ (deg)

(d) Feedback FKF - error (deg)

Figure 12: State Comparison Between Federated Kalman Filter, Single IMU run, and Motion Tracker

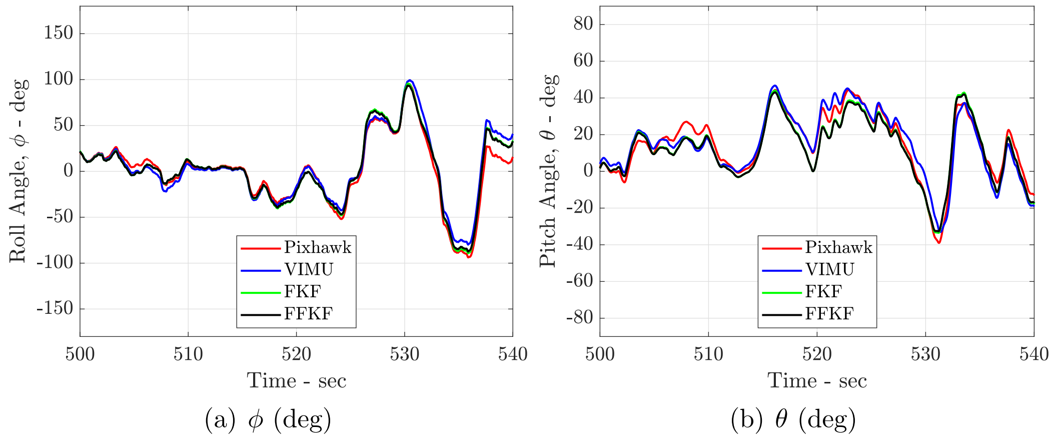


Figure 13: State Comparison Between Pixhawk State Estimation, and Multi-IMU State Estimation

4.2.2 Performance in flight test

Now to see how all the state estimation perform in real flight condition the implemented frameworks were put on am radio controlled (E-Flite mpd Commander) aircraft. The performance of multi-IMU estimation cannot be directly measured quantitatively and hence, we use a qualitative comparison against single-IMU estimation. The single-IMU estimation was performed on Pixhawk at 200Hz and data for 5 IMUs were recorded in the same configuration as mentioned above at 200Hz.

The flight was performed at ambient condition with wind speeds ranging at about 8-10knots. This condition was perfect to test attitude estimator as it would result in a good noise characteristics. The flight trajectory was chosen at random by pilot with covering all ranges of attitude in mind. A small cut of flight is show in Fig. IV.13(a)- IV.13(b)

As a quantitative comparison of all the estimation framework is not trivial, we do a qualitative analysis with attitude estimation in a consistent manner over all flight manuver in mind. Based on the flight test mentioned above, we can observe that all the estimators discussed in this paper perform well compared to Pixhawk system.

Table 2: Measured noise characteristics.

Parameter	Source	Variable	1σ noise
Position	Typical GPS	x, y, z	5m
Erientation	Pixhawk static testing	ϕ, θ, ψ	1°
Airspeed	Typical Pixhawk	u, v, w	0.05 m/s
Ground speed	Typical GPS	$\dot{x}, \dot{y}, \dot{z}$	0.5 m/s
Acceleration	Pixhawk static testing	$a_{x,b}, a_{y,b}, a_{z,b}$	0.1 m/s
Angular rate	Pixhawk static testing	p, q, r	0.1 rad/s

4.3 Tornado wind field simulation setup

For the simulation of proposed model a six degree of freedom model was conducted similar to that of outlined by barton [7]. The aircraft model is based on RC commander aircraft flying at constant velocity of 20 m/s. The flight path and the attitude of the aircraft are simulated based on a fixed trajectory. An on board autopilot module is assumed running the wind estimator derived earlier. GPS measurement are simulated to give aircraft position and airspeed. Aircraft orientation is obtained using the measurements provided by rate gyro and on board magnetometer. The autopilot is assumed to run at about 200Hz (to be consistent with pixhawk) and GPS measurements are obtained at about 10Hz. The noise and uncertainty characteristics for each state is obtained using long term (approx 12hr) static testing of Pixhawk4 and all the values are described in Table2. One more thing to note is that due to the physical limitation of the aircraft the tornado model uses a lowered $V_{\theta,max}$ of 15 m/s.

Gust-free simulation

Without wind gust the only input to the six degree of freedom aircraft simulator is the wind speed velocities obtained through tornado model, and a random flight path taken by the aircraft.

Gust-on simulation - Dryden models

For more realistic simulation a gust model is included with parameters (wind speeds) obtained from tornado vortex models. Gusts fields are modeled using frozen Dryden turbulence model. Thus the wind speed due to gusts are represented as:

$$\mathbf{w} = \mathbf{w}_0 + \sum_{n=1}^{N} \mathbf{a}_n sin(\Omega_n s + \phi_n) \tag{4.3.1}$$

where, $\mathbf{w} = [w_x, w_y, w_z]$, and s is the motion along the path. Now, for the Dryden gust model the power spectral density is defined as

$$\begin{aligned} \Phi_u(\Omega) &= \sigma_u^2 \frac{2L_u}{\pi} \frac{1}{1+(L_u\omega)^2} \\ \Phi_w(\Omega) &= \sigma_w^2 \frac{L_w}{\pi} \frac{1+3(L_w\omega)^2}{(1+(L_u\omega)^2)^2} \end{aligned} \tag{4.3.2}$$

For low altitudes (below 300m), the length scale of the vertical gust is Lw = h and the turbulence intensity is $\sigma_w = 0.1{*}w_6$, where w_6 is the wind speed at 6 m altitude. Horizontal gust length scale and intensity are related to the vertical gust scale and intensity by following equations where h is in m.

$$\begin{aligned} \frac{L_u}{L_w} &= \frac{1}{(0.177+0.000823h)^{1.2}} \\ \frac{\sigma_u}{\sigma_w} &= \frac{1}{(0.177+0.000823h)^{0.4}} \end{aligned} \tag{4.3.3}$$

The amplitude of a sinusoidal gust in equation4.3.1 is computed as:

$$\mathbf{a}_n = \sqrt{\Delta\Omega_n \Phi(\omega_n)}$$

The other parameters of the simulation were as from the gust free condition.

4.3.1 Tornado wind speed estimation simulation

GPS path is simulated in the wind field is shown in Fig 14. The absolute attitude obtained through the estimator against the true are shown in the Fig 16 - 18. The wind estimation obtained using the method is shown in the Fig 15 and the error plot for wind estimation is shown in Fig 19. The root mean squared value of wind error are obtained to be about 1.76 m/s and root mean squared error values for all parameters are shown in Table 3. For the simulation run at low wind speed the error in wind speed estimation is comparatively high for the use of direct methods in application of tornado wind speed estimation, and this could be because the direct wind estimation method uses the GPS velocity and compares it to velocity obtained by airspeed sensor and hence, is limited by the frequency of GPS data obtained. The results obtain show that for a better tornado wind speed estimation we need a better wind estimation routine for the same hardware or a different method that does not depend on the GPS measurement data and uses other high speed measurement data available like that of accelerometer and gyroscope.

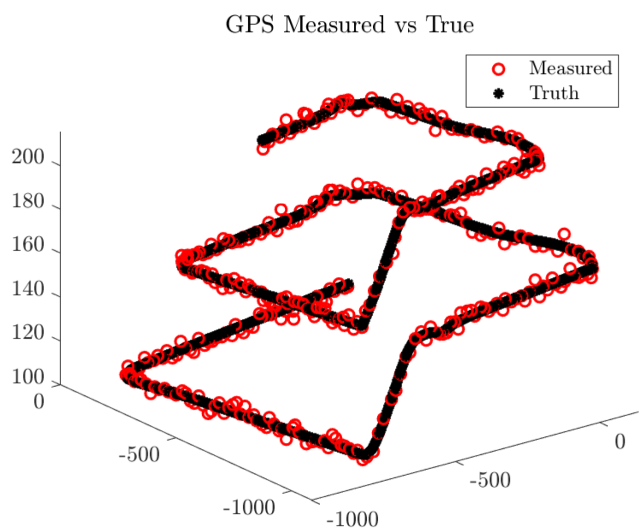


Figure 14: Flight Trajectory taken by the aircraft in simulated wind field for both no gust and gust condition.

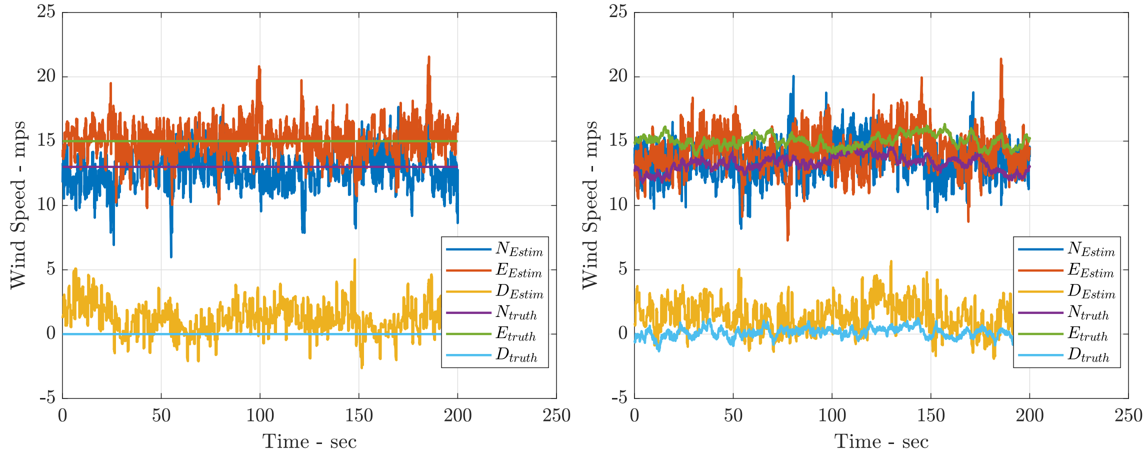


Figure 15: Wind Estimation for both no gust [left] condition and gust [right].

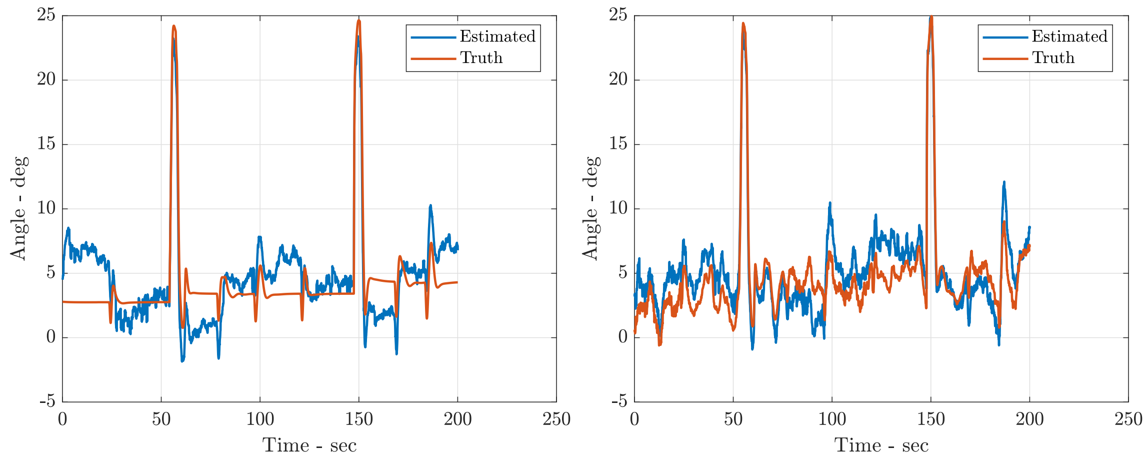


Figure 16: Pitch Estimation for both gust [right] and no gust [left] condition

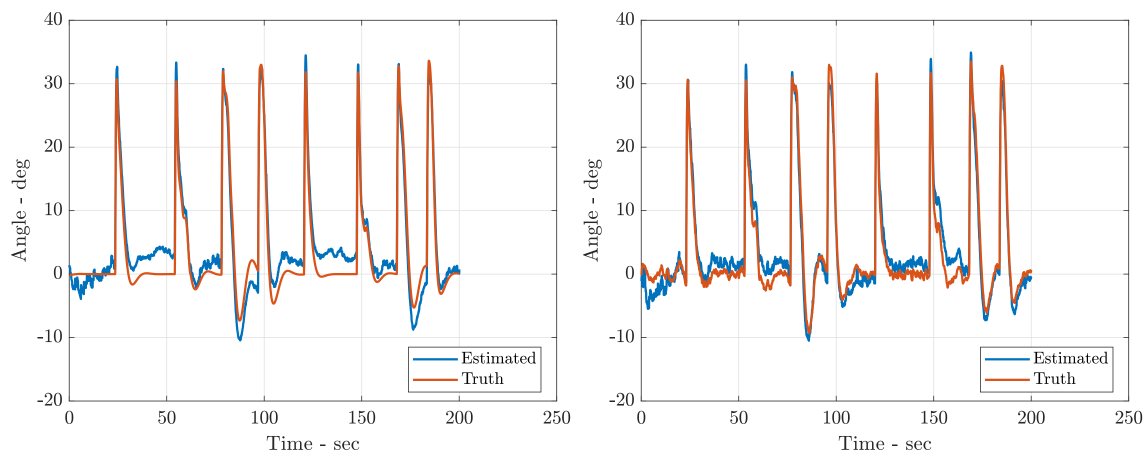


Figure 17: Roll Estimation for both gust [right] and no gust [left] condition

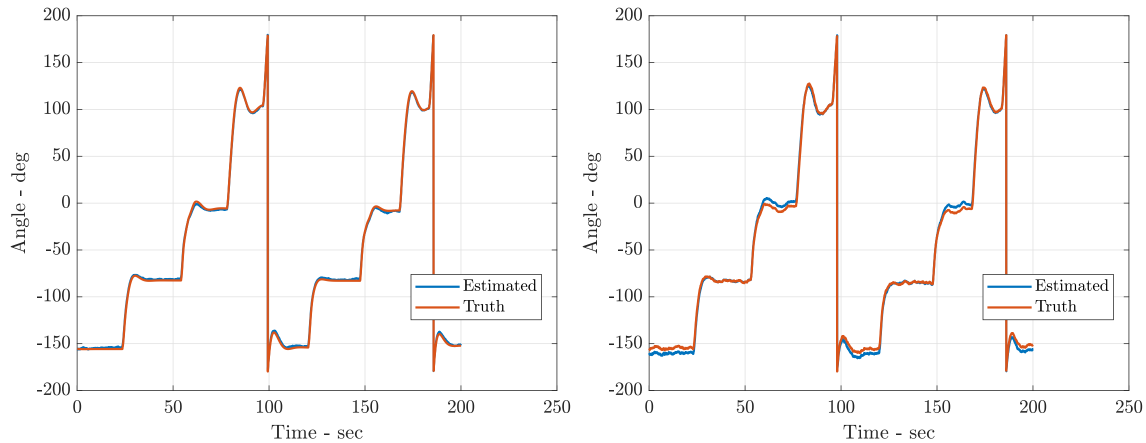


Figure 18: Yaw Estimation for both gust [right] and no gust [left] condition

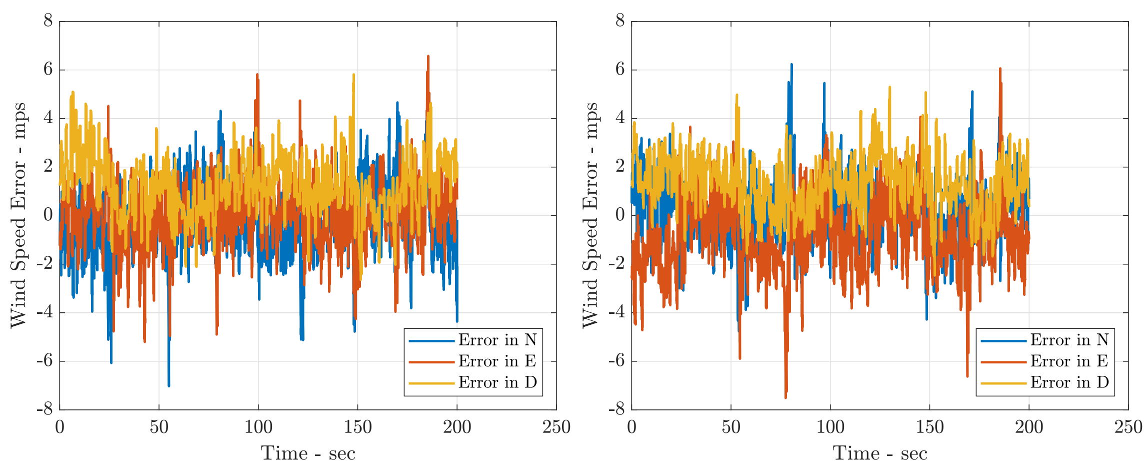


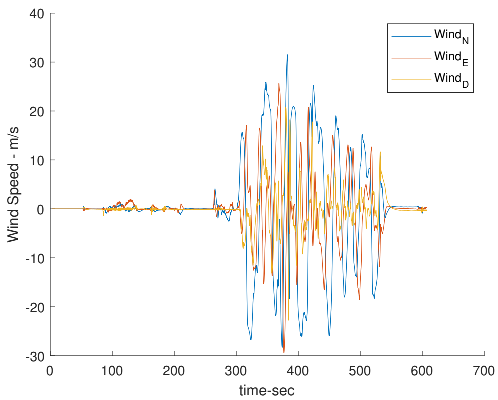
Figure 19: Wind Estimation Error for both gust [right] and no gust [left] condition

Table 3: RMS Error of estimated states and disturbances.

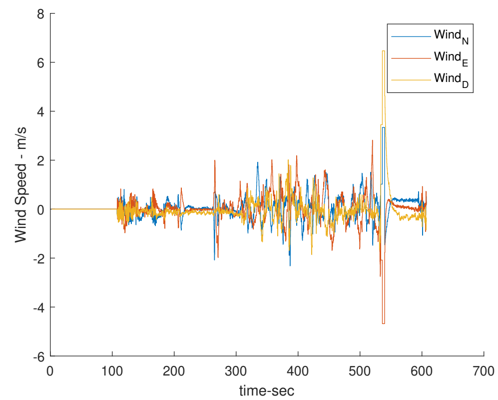
Parameter	Units	With Gust	Without Gust
Pitch	deg	2.2891	1.8017
Roll	deg	2.3829	1.9841
Yaw	deg	5.8676	8.8836
Wind estimate in N direction	m/s	1.4015	1.3161
Wind estimate in E direction	m/s	1.3008	1.2159
Wind estimate in D direction	m/s	1.7634	1.7598

Flight test for Direct and IMU based wind estimation

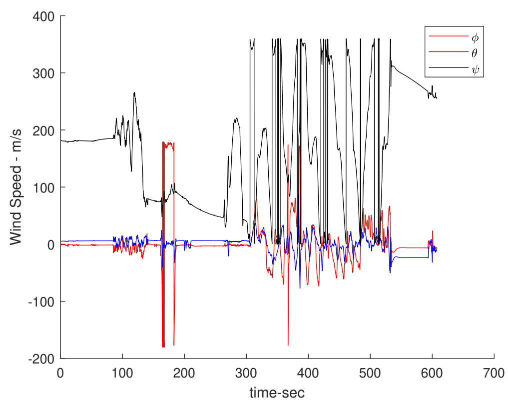
The method outlined in the Section 3.3.1 and Section 3.3.1 is implemented in the same experimental setup as Section 4.2.2. This flight test was done to get preliminary tests for wind field estimation.



(a) Wind Estimation using IMU



(b) Wind Estimation using GPS



(c) Attitude of Aircraft during wind estimation

Figure 20: Wind estimation routine for flight test.

Fig. 20 shows the results of the wind estimation results. We cab observe that the wind estimation using direct method is very reasonable where as the wind estimation using IMU method gives highly dynamic wind, and this difference can be attributed to inaccurate aircraft model.

CHAPTER V

Conclusion

In this study, the idea of fusing multiple-imu to improve the accuracy and reliability of the state estimation for UAS was investigated. Four state estimation fusion methods were tested with two different local attitude estimators. The state estimators were than implemented on data collected from 5 different IMU's (2 from Navio2 on Raspberry Pi, and 3 from Pixhawk 2.1). The estimators were tested against the reference data obtained using motion tracker system (OptiTracker).

The actual experimental results demonstrated that feedback federated Kalman filter with attitude estimation using Bortz equation had the best performance as measured when compared to all the other methods. Table 4 shows the root mean square error for all the implemented runs using the 5 IMUs. Moreover, the estimators were tested independently using different IMUs as as sensors to give the results shown in Fig. 21. The results consistently indicate that Federated Kalman filter has the best improvement in state estimation accuracy no matter the number on IMU's employed.

Table 4: Measured State Error.

Parameter	**FFKF**	**FKF**	**vIMU**	**AKF**
Error in ϕ	4.0229	4.793	6.180	15.748
Error in θ	1.168	1.190	1.371	7.100
Error in ψ	5.8813	5.878	19.102	19.892

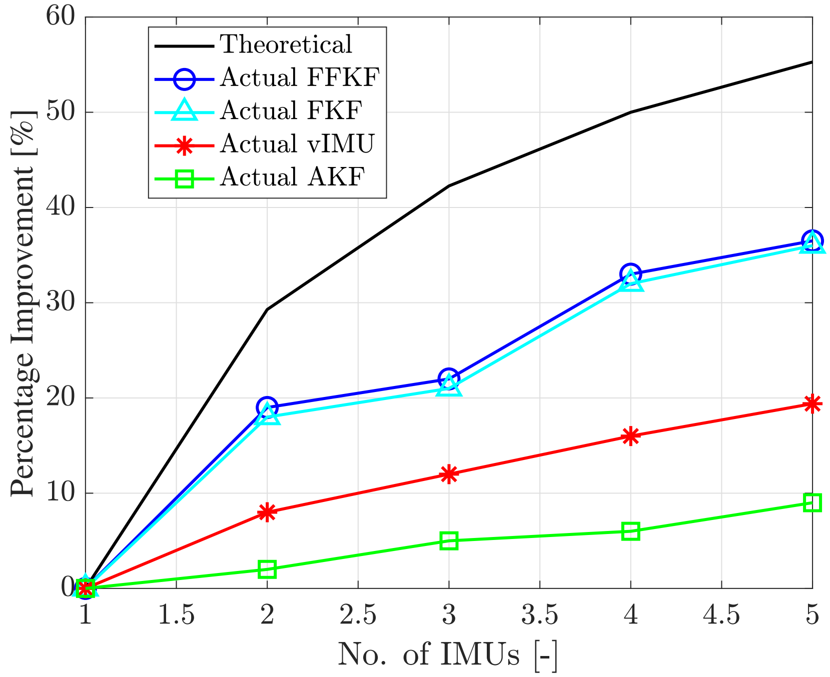


Figure 21: Idealized and experimentally implemented multi-IMU estimator performance for varying approaches.

This study also introduces an approach for wind field estimate validation in a simulated tornado, defines a procedure of model comparison and validation of tornado using hardware-in-loop simulation. The approach consists of simulating tornado wind speeds using an idealized tornadic model, using spatiotemporal interpolation to obtain a local measurement, and implementing the traditional ”direct” wind field estimator. Estimated and true wind speeds were compared quantify performance of the wind estimation method. The results suggest a need for improvements in current wind estimation routines. Although the study lays a strong base for the purpose of tornado model validation using real data and real flight implementation.

5.1 Future work

The results obtained in this study clearly show the potential of employing multi-IMU based state estimation to not only improve the accuracy of the estimates and also to add redundancies in the system. Future work includes the extension of the studies attitude estimation to include on board wind speed estimation for UAS and having multiple decentralized agents to do wind field estimation.

5.1.1 Wind field flight test

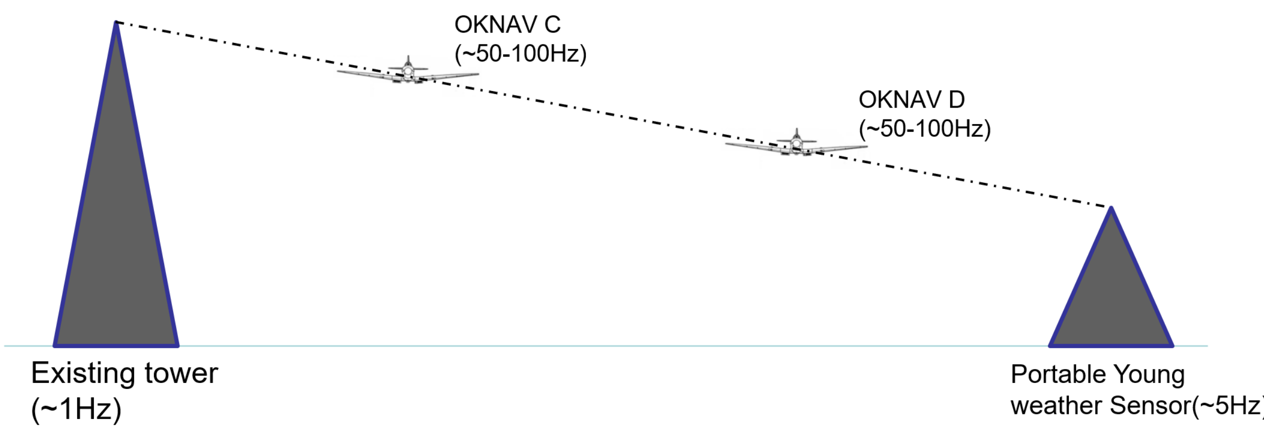


Figure 22: Wind Field Estimation setup in field test

To test how well the wind field estimation would work as outline in the simulation, we need a experimental layout as shown in Fig. 22. This layout is chosen as two high accuracy wind speed estimating tower would act as a reference to validate the wind speed estimation onborf UAS and also act as a accurate boundary condition for wind field estimation.

This experimental layout works for multi rate system as the wind field estimation works in spatiotemporal dynamic system. Actual flight test would lay a ground work for wind field estimation using multiple UAS as decentralized agents to do high accuracy wind field estimation.

REFERENCES

[1] *Ieee standard specification format guide and test procedure for linear, single-axis, pendulous, analog torque balance accelerometer*, IEEE Std 337-1972 (1972), 1–48.

[2] *Ieee standard specification format guide and test procedure for coriolis vibratory gyros*, IEEE Std 1431-2004 (2004), 1–78.

[3] Robert L Ash and Irfan R Zardadkhan, *Non-equilibrium behavior of large-scale axial vortex cores*, AIP advances **11** (2021), no. 2, 25320–025320–6 (eng).

[4] Alyssa Shearon Avery, *Design and development of a severe storm research uas*, 2015.

[5] J. B. Bancroft, *Multiple inertial measurement unit fusion for pedestrian navigation*, 2011.

[6] J. B. Bancroft and G. Lachapelle, *Data fusion algorithms for multiple inertial measurement units*, Sensors (Basel, Switzerland) **11** (2011), no. 7, 6771–6798 (eng).

[7] J D Barton, *Fundamentals of small unmanned aircraft flight*, JOHNS HOPKINS APL TECHNICAL DIGEST **2** (2012), no. 31, 132, 149.

[8] Y. Beaudoin, A. Desbiens, E. Gagnon, and R. Landry, *Satellite launcher navigation with one versus three imus: Sensor positioning and data fusion model analysis*, Sensors **18** (2018), no. 6, 1872.

[9] ———, *Satellite launcher navigation with one versus three imus: Sensor positioning and data fusion model analysis*, Sensors **18** (2018), no. 6.

[10] Y. Bezkorovainyi and O. Sushchenko, *Improvement of uav positioning by information of inertial sensors*, 2018 IEEE 5th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC), 2018, pp. 123–126.

[11] J. E. Bortz, *A new mathematical formulation for strapdown inertial navigation*, IEEE Transactions on Aerospace and Electronic Systems **AES-7** (1971), no. 1, 61–66 (eng).

[12] A. Brown and M. A. Sturza, *The effect of geometry on integrity monitoring performance*, ION GPS (1990), 121–129.

[13] N. A. Carlson, *Federated square root filter for decentralized parallel processors*, IEEE Transactions on Aerospace and Electronic Systems **26** (1990), no. 3, 517–525.

[14] N. A Carlson and M. P. Berarducci, *Federated kalman filter simulation results*, Navigation **41** (1994), no. 3, 297–322.

[15] I. Colomina, M. Giménez, J. Rosales, M. Wis, A.Gómez, and P. Miguelsanz, *Redundant imus for precise trajectory determination*, Institute of Geomatics (2004).

[16] K. Eckenhoff and and G. Huang P. Geneva, *Sensor-failure-resilient multi-imu visual-inertial navigation*, 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 3542–3548.

[17] Jack Elston, Brian Argrow, Adam Houston, and Eric Frew, *Design and validation of a system for targeted observations of tornadic supercells using unmanned aircraft*, 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 101–106.

[18] E. W Frew, J Elston, B Argrow, A Houston, and E Rasmussen, *Sampling severe local storms and related phenomena: Using unmanned aircraft systems*, IEEE robotics & automation magazine **19** (2012), no. 1, 85–95 (eng).

[19] C. Gao, G. Zhao, J. Lu, and S. Pan, *Decentralised moving-horizon state estimation for a class of networked spatial-navigation systems with random parametric uncertainties and communication link failures*, IET Control Theory & Applications **9** (2015), no. 18, 2666–2677.

[20] M. Givens, C. Coopmans, and R. Christensen, *An estimation-domain approach to mems multi-imu fusion for suas*, 2019 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2019, pp. 1048–1053 (eng).

[21] J. González-Rocha, C. A. Woolsey, C. Sultan, and S.J. De Wekker, *Sensing wind from quadrotor motion*, Journal of Guidance, Control, and Dynamics **42** (2019), no. 4, 836–852 (eng).

[22] J. Gross, Y. Gu, M. Rhudy, S. Gururajan, and M. Napolitano, *Flight-test evaluation of sensor fusion algorithms for attitude estimation*, IEEE Transactions on Aerospace and Electronic Systems **48** (2012), no. 3, 2128–2139.

[23] T. R. Kane and L. A. David, *Dynamics: Theory and applications*, McGraw-Hill Book Co., 2005.

[24] C. Kang and P. Chan, *A soft-failure detection and identification algorithm for the integrated navigation system of lunar lander*, Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering **230** (2015).

[25] A. Khosravian, J. Trumpf, R. Mahony, and T. Hamel, *Recursive attitude estimation in the presence of multi-rate and multi-delay vector measurements*, 2015 American Control Conference (ACC), July 2015, pp. 3199–3205.

[26] Jack W. Langelaan, Nicholas Alley, and James Neidhoefer, *Wind field estimation for small unmanned aerial vehicles*, Journal of Guidance, Control, and Dynamics **34** (2011), no. 4, 1016–1030.

[27] Miodrag Lovric (ed.), *International encyclopedia of statistical science*, Springer Berlin Heidelberg, 2011.

[28] J. C. McMillan, J. S. Bird, and D. A. ARDEN, *Techniques for soft-failure detection in a multisensor integrated system*, NAVIGATION **40** (1993), no. 3, 227–248.

[29] Robert C. Nelson, *Flight stability and automatic control*, McGraw-Hill, New York, 1989 (eng).

[30] Xiaoji Niu, Qingjiang Wang, You Li, Qingli Li, and Jingnan Liu, *Using inertial sensors in smartphones for curriculum experiments of inertial navigation technology*, Education Sciences **5** (2015), 26–46.

[31] C. I. Oliver, *Chapter 6 - functions of random variables*, Fundamentals of Applied Probability and Random Processes (Second Edition) (Oliver C. Ibe, ed.), Academic Press, Boston, second edition ed., 2014, pp. 185–223.

[32] T. Ozyagcilar, *Calibrating an ecompass in the presence of hard and soft-iron interference*, (2015).

[33] U. Patel and I. Faruque, *Multi-imu unmanned aerial system state estimation: frameworks and performance comparison.*

[34] M. Pavel and P. Ilya, *Mems-based non-orthogonal redundant inertial measurement unit for miniature navigation systems*, 2015 International Siberian Conference on Control and Communications (SIBCON) (2015), 1–3.

[35] M. E. Pittelkau, *Rotation vector in attitude estimation*, Journal of Guidance, Control, and Dynamics **26** (2003), no. 6, 855–860 (eng).

[36] Robert. H. Rogne, Torleiv. H. Bryne, Thor. I. Fossen, and Tor. A. Johansen, *Redundant MEMS-Based Inertial Navigation Using Nonlinear Observers*, Journal of Dynamic Systems, Measurement, and Control **140** (2018), no. 7, 071001.

[37] Andrew Levi Ross, *Design and development of a tornado intercept unmanned aerial vehicle*, 2019.

[38] P. G. Saffman, *Vortex dynamics*, Cambridge monographs on mechanics and applied mathematics, Cambridge University Press, Cambridge ;, 1992 (eng).

[39] B. M. Scheninger, J. Hutton, and J. C. McMillan, *Low cost inertial/gps integrated position and orientation system for marine applications*, Aerospace and Electronic Systems Magazine, IEEE **12** (1997), 15 – 19.

[40] RS Scorer, *Origin of cyclones*, Sci. J **2** (1966), no. 3, 46–52.

[41] I. Skog, J. Nilsson, and P. Handel, *An open-source multi inertial measurement unit (mimu) platform*, 2014 International Symposium on Inertial Sensors and Systems (ISISS) (2014), 1–4.

[42] I. Skog, J. Nilsson, and P. Händel, *Pedestrian tracking using an imu array*, 2014 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2014, pp. 1–4.

[43] STMicroelectronics, *Ultra-compact high-performance ecompass module: 3d accelerometer and 3d magnetometer*, 2013, Rev. 2.

[44] STMicroelectronics, *inemo inertial module: 3d accelerometer, 3d gyroscope, 3d magnetometer*, 2015, Rev. 3.

[45] M. A. Sturza, *Navigation system integrity monitoring using redundant measurements*, Navigation **35** (1988), no. 4, 483–501.

[46] S. Sukkarieh, P. Gibbens, B. Grocholsky, K. Willis, and H. Durrant-Whyte, *A low-cost, redundant inertial measurement unit for unmanned air vehicles*, The International Journal of Robotics Research **19** (2000), no. 11, 1089–1103.

[47] M. Tanenhaus, T. Geis, D. Carhoun, and A. Holland, *Accurate real time inertial navigation device by application and processing of arrays of mems inertial sensors*, IEEE/ION Position, Location and Navigation Symposium, 2010, pp. 20–26.

[48] TDK, *Mpu-9250 product specification*, 2016, Rev. 1.1.

[49] A. Waegli, S. Guerrier, and J. Skaloud, *Redundant mems-imu integrated with gps for performance assessment in sports*, 2008 IEEE/ION Position, Location and Navigation Symposium, 2008, pp. 1260–1268.

[50] Y. Yang and W. Gao, *An optimal adaptive kalman filter*, Journal of geodesy **80** (2006), no. 4, 177–183 (eng).

[51] A. M Zamorzaev and A. F Palistrant, *Rotations, quaternions and double groups by s. altmann*, Acta Crystallographica Section A **44** (1988), no. 4, 622–622 (eng).

[52] J. Zhang and J. Hongbing, *Distributed multi-sensor particle filter for bearings-only tracking*, International Journal of Electronics **99** (2012), no. 2, 239–254.

APPENDICES

Calibration

Magnetometer calibration for multi-IMU was based on the routine outlined by Ozyagcilar T [32], which corrects for hard and soft iron interference. The calibration process consists of fitting a set of ten model parameters to the magnetometer measurements. four parameters model the hard-iron offset, six model the soft-iron matrix and one models the geomagnetic field strength.

In the multi-IMU case, the magnetometers used in the experiment are not identical and can have biases and offsets. After the calibration parameters were identified, a normalization was implemented to adjust for individual magnetometer scale variations. Fig. 23 shows an example of the magnetometer reading during a rotation about all axes before calibration and Fig. 24 shows the same calibration loop corrected for both hard and soft interference.

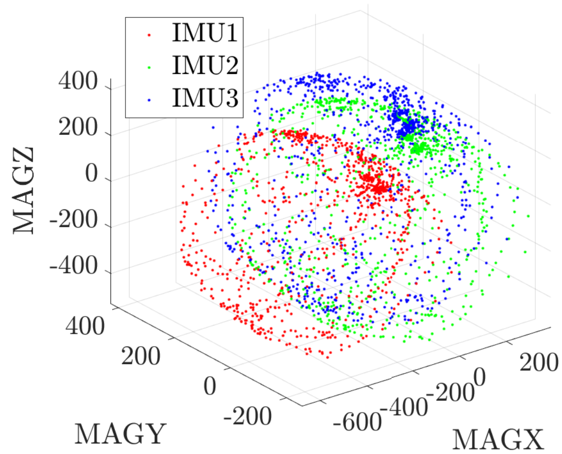


Figure 23: Uncalibrated magnetometer.

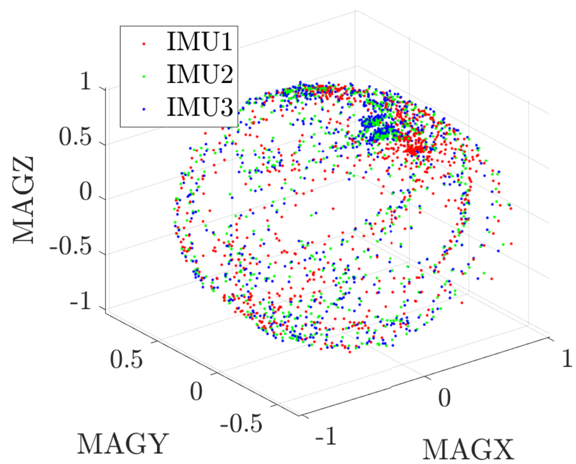


Figure 24: Calibrated magnetometer.

Airplane

Kinematic equation as defined in the book Stability and Control by Robert Nelson [29].

$$
\begin{aligned}
X - mgS_\theta &= m(\dot{u} + qw - rv)\\
Y + mgC_\theta S_\Phi &= m(\dot{v} + ru - pw)\\
Z + mgC_\theta C_\Phi &= m(\dot{w} + pv - qu)
\end{aligned}
$$

Force equations

$$
\begin{aligned}
L &= I_x\dot{p} - I_{xz}\dot{r} + qr(I_z - I_y) - I_{xz}pq\\
M &= I_y\dot{q} + rp(I_x - I_z) + I_{xz}(p^2 - r^2)\\
N &= -I_{xz}\dot{p} + I_z\dot{r} + pq(I_y - I_x) + I_{xz}qr
\end{aligned}
$$

Moment equations

$$
\begin{aligned}
p &= \dot{\Phi} - \dot{\psi}S_\theta\\
q &= \dot{\theta}C_\Phi + \dot{\psi}C_\theta S_\Phi\\
r &= \dot{\psi}C_\theta C_\Phi - \dot{\theta}S_\Phi
\end{aligned}
$$

Body angular velocities in terms of Euler angles and Euler rates

$$
\begin{aligned}
\dot{\theta} &= qC_\Phi - rS_\Phi\\
\dot{\Phi} &= p + qS_\Phi T_\theta + rC_\Phi T_\theta\\
\dot{\psi} &= (qS_\Phi + rC_\Phi)\sec\theta
\end{aligned}
$$

Euler rates in terms of Euler angles and body angular velocities

Velocity of aircraft in the fixed frame in terms of Euler angles and body velocity components

$$
\begin{bmatrix} \dfrac{dx}{dt} \\ \dfrac{dy}{dt} \\ \dfrac{dz}{dt} \end{bmatrix}
=
\begin{bmatrix}
C_\theta C_\psi & S_\Phi S_\theta C_\psi - C_\Phi S_\psi & C_\Phi S_\theta C_\psi + S_\Phi S_\psi \\
C_\theta S_\psi & S_\Phi S_\theta S_\psi + C_\Phi C_\psi & C_\Phi S_\theta S_\psi - S_\Phi C_\psi \\
-S_\theta & S_\Phi C_\theta & C_\Phi C_\theta
\end{bmatrix}
\begin{bmatrix} u \\ v \\ w \end{bmatrix}
$$

Allan Variance

The Allan deviation plot is a method of graphing the various error sources of a time-series of data on a single plot. The method was first introduced by David Allan in 1966 to measure the frequency stability of clocks and oscillators. The technique is useful for inertial navigation systems since it allows both the angle/velocity random walk and bias stability of the sensors to be determined in a single plot.

To compute the Allan deviation for a time series of data x_i, begin by splitting the data series into bins of size n where N is the number of resulting bins. Let y_i be the average of bin i where $i = 1, \ldots, N$. The Allan variance of x_i is given by

$$\sigma^2(\tau) = \frac{1}{2(N-1)} \sum_{1}^{N-1} (x_{i+1} - xi)^2$$

where τ is the time constant for consecutive samples in x_i. The Allan deviation then is found by taking the square root of the Allan variance. For interpretation of the Allan deviation plot, please refer to [[2], [1]]. A sample of Allan variation plot is shown 25 and with corresponding noise and slopes.

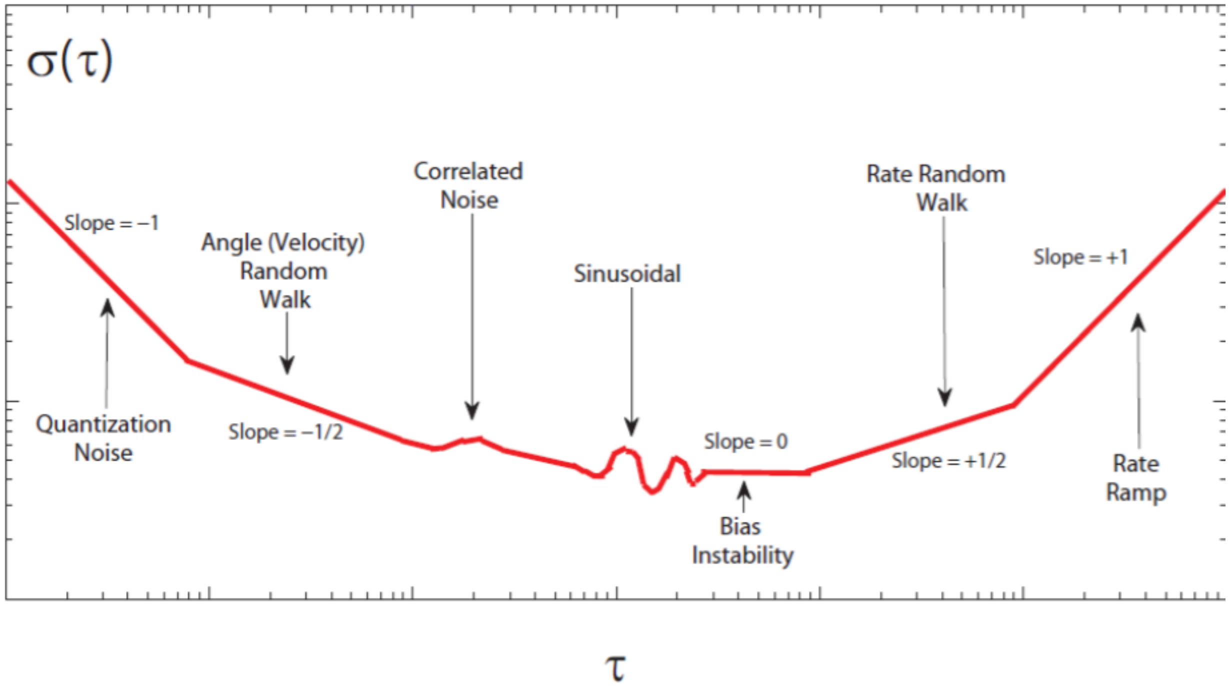


Figure 25: Sample Allan deviation plot as described in [30].

CODES

```
function quaterion = EulToQuat(Euler)

quaterion = zeros(4,1);

Euler     = Euler * 0.5;
cosPhi    = cos(Euler(1));
sinPhi    = sin(Euler(1));
cosTheta  = cos(Euler(2));
sinTheta  = sin(Euler(2));
cosPsi    = cos(Euler(3));
sinPsi    = sin(Euler(3));

quaterion(1,1) = (cosPhi*cosTheta*cosPsi + sinPhi*sinTheta*sinPsi);
quaterion(2,1) = (sinPhi*cosTheta*cosPsi - cosPhi*sinTheta*sinPsi);
```

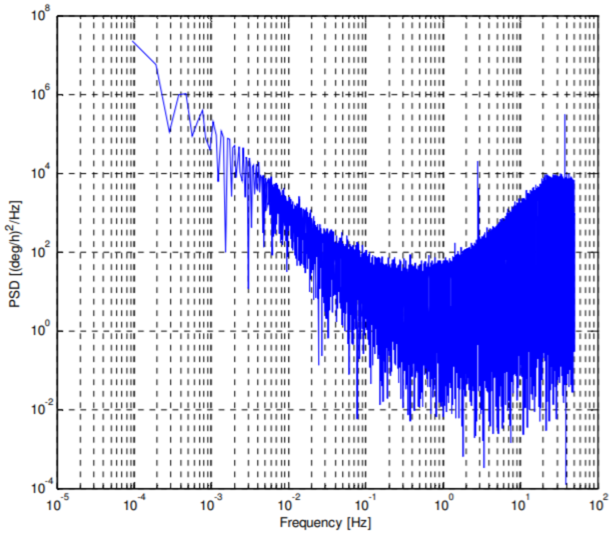


Figure 26: Single-axis-gyro Power Spectral Density(PSD)

```
quaterion(3,1) = (cosPhi*sinTheta*cosPsi + sinPhi*cosTheta*sinPsi);
quaterion(4,1) = (cosPhi*cosTheta*sinPsi - sinPhi*sinTheta*cosPsi);

return;
```

```
function quaternion = normQuat(quaternion)

quatMag = sqrt(quaternion(1)^2 + quaternion(2)^2 + quaternion(3)^2 + ...
    quaternion(4)^2);
quaternion(1:4) = quaternion / quatMag;
```

```
function Tbn = Quat2Tbn(quat)

q0 = quat(1);
q1 = quat(2);
q2 = quat(3);
q3 = quat(4);

```

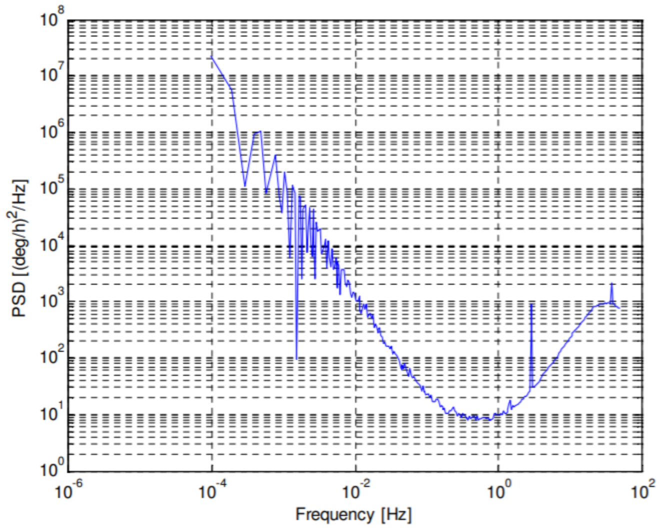


Figure 27: Single-axis-gyro PSD with frequency averaging

```
Tbn = [q0^2 + q1^2 - q2^2 - q3^2, 2*(q1*q2 - q0*q3), 2*(q1*q3 + q0*q2); ...
      2*(q1*q2 + q0*q3), q0^2 - q1^2 + q2^2 - q3^2, 2*(q2*q3 - q0*q1); ...
      2*(q1*q3-q0*q2), 2*(q2*q3 + q0*q1), q0^2 - q1^2 - q2^2 + q3^2];
```

```
function q_out = QuatDivide(qin1,qin2)

q0 = qin1(1);
q1 = qin1(2);
q2 = qin1(3);
q3 = qin1(4);

r0 = qin2(1);
r1 = qin2(2);
r2 = qin2(3);
r3 = qin2(4);

q_out(1,1) = (qin2(1)*qin1(1) + qin2(2)*qin1(2) + qin2(3)*qin1(3) + ...
    qin2(4)*qin1(4));
q_out(2,1) = (r0*q1 - r1*q0 - r2*q3 + r3*q2);
```

```
q_out(3,1) = (r0*q2 + r1*q3 - r2*q0 - r3*q1);
q_out(4,1) = (r0*q3 - r1*q2 + r2*q1 - r3*q0);
```

```
function quatOut = QuatMult(quatA,quatB)

quatOut = [quatA(1)*quatB(1)-quatA(2:4)'*quatB(2:4); ...
    quatA(1)*quatB(2:4) + quatB(1)*quatA(2:4) + ...
    cross(quatA(2:4),quatB(2:4))];
```

```
% Convert from a quaternion to a 321 Euler rotation sequence in radians

function Euler = QuatToEul(quat)

Euler = zeros(3,1);

Euler(1) = atan2(2*(quat(3)*quat(4)+quat(1)*quat(2)),  quat(1)*quat(1) ...
    - quat(2)*quat(2) - quat(3)*quat(3) + quat(4)*quat(4));
Euler(2) = -asin(2*(quat(2)*quat(4)-quat(1)*quat(3)));
Euler(3) = atan2(2*(quat(2)*quat(3)+quat(1)*quat(4)),  quat(1)*quat(1) ...
    + quat(2)*quat(2) - quat(3)*quat(3) - quat(4)*quat(4));
```

```
function quaternion = RotToQuat(rotVec)

vecLength = sqrt(rotVec(1)^2 + rotVec(2)^2 + rotVec(3)^2);

if vecLength < 1e-6
  quaternion = [1;0;0;0];
else
  quaternion = [cos(0.5*vecLength); rotVec/vecLength*sin(0.5*vecLength)];
```

```
end
```

```
%% Start
close all;
clear;
clc;

%% Choose Fligth No
Test_num = 2;
%% Options
magcal = 0;
Plot_Floder_Name = append('Final_Plots_1117_',num2str(Test_num));
fig_name = 'fig_';
saveplots = 1;
MasterFusion = 1; % 0 For simple average % 1 For FKF

% Set the expected declination
measDec = -30*pi/180;
%% Include Path
addpath('Calibration')
addpath('SkinnyC')
addpath('AttErrVecMathExample')
addpath('Common')
addpath('quaternion_library')
addpath('Flight_Data_04182021')
load(append('Test',num2str(Test_num),'.mat'));

%% MagneticCalibration
i = 1;
if magcal ==1
    plotcal = 1;
```

```
    magcalrun % Creates and saves new CalVal
    clearvars -except Plot_Floder_Name fig_name saveplots fig ...
        MasterFusion i
end

load('Calibration/CalVal')

startDelayTime = 0; % number of seconds to delay filter start (used to ...
    sIMU2late in-flight restart)
dt = 1/200;
indexLimit = length(IMU3);
MAGIndexlimit = length(MAG3);

%% Set the RUN

% State Log
statesLog = zeros(10,indexLimit);
statesLog2 = zeros(10,indexLimit);
statesLog3 = zeros(10,indexLimit);
statesLog4 = zeros(10,indexLimit);
statesLog5 = zeros(10,indexLimit);
statesLogm = zeros(10,indexLimit);
statesLogv = zeros(10,indexLimit);

% Euler Log
eulLog   = zeros(4,indexLimit);
eulLog2   = zeros(4,indexLimit);
eulLog3   = zeros(4,indexLimit);
eulLog4   = zeros(4,indexLimit);
eulLog5   = zeros(4,indexLimit);
eulLogm   = zeros(4,indexLimit);
eulLogv   = zeros(4,indexLimit);
```

```
% Quaternion Log
quatLog    = zeros(4,indexLimit);
quatLog2    = zeros(4,indexLimit);
quatLog3    = zeros(4,indexLimit);
quatLog4    = zeros(4,indexLimit);
quatLog5    = zeros(4,indexLimit);
quatLogm    = zeros(4,indexLimit);
quatLogv    = zeros(4,indexLimit);

% Velocity Innov Log

velInnovLog = zeros(4,indexLimit);
velInnovLog2 = zeros(4,indexLimit);
velInnovLog3 = zeros(4,indexLimit);
velInnovLog4 = zeros(4,indexLimit);
velInnovLog5 = zeros(4,indexLimit);
velInnovLogm = zeros(4,indexLimit);
velInnovLogv = zeros(4,indexLimit);

% Angular Error Log
angErrLog = zeros(2,indexLimit);
angErrLog2 = zeros(2,indexLimit);
angErrLog3 = zeros(2,indexLimit);
angErrLog4 = zeros(2,indexLimit);
angErrLog5 = zeros(2,indexLimit);
angErrLogm = zeros(2,indexLimit);
angErrLogv = zeros(2,indexLimit);

% Measured Velcoity Log
measVelLog = zeros(3,indexLimit);
measVelLog2 = zeros(3,indexLimit);
```

```
measVelLog3 = zeros(3,indexLimit);
measVelLog4 = zeros(3,indexLimit);
measVelLog5 = zeros(3,indexLimit);
measVelLogm = zeros(3,indexLimit);
measVelLogv = zeros(3,indexLimit);

% Declination Log
decInnovLog = zeros(2,MAGIndexlimit);
decInnovLog2 = zeros(2,MAGIndexlimit);
decInnovLog3 = zeros(2,MAGIndexlimit);
decInnovLog4 = zeros(2,MAGIndexlimit);
decInnovLog5 = zeros(2,MAGIndexlimit);
decInnovLogm = zeros(2,MAGIndexlimit);
decInnovLogv = zeros(2,MAGIndexlimit);

% Var Inov Log
velInnovVarLog = velInnovLog;
decInnovVarLog = decInnovLog;
velInnovVarLog2 = velInnovLog;
decInnovVarLog2 = decInnovLog;
velInnovVarLog3 = velInnovLog;
decInnovVarLog3 = decInnovLog;
velInnovVarLog4 = velInnovLog;
decInnovVarLog4 = decInnovLog;
velInnovVarLog5 = velInnovLog;
decInnovVarLog5 = decInnovLog;
velInnovVarLogv = velInnovLog;
decInnovVarLogv = decInnovLog;

%% Initialization Run

% Init State
```

```
states = zeros(9,1);
states2 = zeros(9,1);
states3 = zeros(9,1);
states4 = zeros(9,1);
states5 = zeros(9,1);
statesm = zeros(3,1);
statesv = zeros(9,1);

% Init Quat
quatm = [1;0;0;0];
quat = [1;0;0;0];
quat2 = [1;0;0;0];
quat3 = [1;0;0;0];
quat4 = [1;0;0;0];
quat5 = [1;0;0;0];
quatv = [1;0;0;0];

% Init Transformation Matrix
Tbn = Quat2Tbn(quat);
Tbn2 = Quat2Tbn(quat2);
Tbn3 = Quat2Tbn(quat3);
Tbn4 = Quat2Tbn(quat4);
Tbn5 = Quat2Tbn(quat5);
Tbnv = Quat2Tbn(quatv);

% Init Acceleration
initAccel(:) = mean(IMU(1:10,6:8));
initAccel2(:) = mean(IMU2(1:10,6:8));
initAccel3(:) = mean(IMU3(1:10,6:8));
initAccel4(:) = mean(IMU4(1:10,6:8));
initAccel5(:) = mean(IMU5(1:10,6:8));
initAccelv(:) = (initAccel+initAccel2+initAccel3+initAccel4+initAccel5)/5;
```

```
% Use averaged accel readings to align tilt
quat = AlignTilt(quat,initAccel);
quat2 = AlignTilt(quat2,initAccel2);
quat3 = AlignTilt(quat3,initAccel3);
quat4 = AlignTilt(quat4,initAccel4);
quat5 = AlignTilt(quat5,initAccel5);
quatv = AlignTilt(quatv,initAccelv);

% define the state covariances
Sigma_velNED = 0.5; % 1 sigma uncertainty in horizontal velocity components
Sigma_dAngBias  = 1*pi/180*dt; % 1 Sigma uncertainty in Δ angle bias
Sigma_angErr = 1; % 1 Sigma uncertainty in angular misalignment (rad)


covariance = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariance2 = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariance3 = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariance4 = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariance5 = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariancev = ...
    single(diag([Sigma_angErr*[1;1;1];Sigma_velNED*[1;1;1];Sigma_dAngBias*[1;1;1]].^2));
covariancem = covariance(1:3,1:3)*5;
%% MagCalibration

data = MAG(:,3:5)';
MAG(:,3:5) = (A1*(data-repmat(c1,1,length(MAG))))';
```

```
data2 = MAG2(:,3:5)';
MAG2(:,3:5) = (A2*(data2-repmat(c2,1,length(MAG2))))';

data3 = MAG3(:,3:5)';
MAG3(:,3:5) = (A3*(data3-repmat(c3,1,length(MAG3))))';

%% Main Loop

MAGIndex = 1;
time = 0;
angErr = 0;
headingAligned = 0;
% delay start by a minIMU2m of 10 IMU3 closamples to allow for initial tilt
% alignment delay
% startIndex = max(1,ceil(startDelayTime/dt));
% to deal with the GPS vel
j = 1;
measVel = [0;0;0];
looptime = [];

for index = 1:indexLimit % startIndex:indexLimit

    time=time+dt*1000; % + startIndex*dt*1000;
    % read IMU3 measurements
    angRate = IMU(index,3:5)';
    angRate2 = IMU2(index,3:5)';
    angRate3 = IMU3(index,3:5)';
    angRate4 = IMU4(index,3:5)';
    angRate5 = IMU5(index,3:5)';
    angRatev = (angRate + angRate2 + angRate3 + angRate4 + angRate5)/5;

```

```
    % switch in a bias offset to test the filter
    if (time > +inf)
        angRate = angRate + [1;-1;1]*pi/180;
        angRate2 = angRate2 + [1;-1;1]*pi/180;
        angRate3 = angRate3 + [1;-1;1]*pi/180;
        angRate4 = angRate4 + [1;-1;1]*pi/180;
        angRate5 = angRate5 + [1;-1;1]*pi/180;
        angRatev = angRatev + [1;-1;1]*pi/180;
    end

    accel = IMU(index,6:8)';
    accel2 = IMU2(index,6:8)';
    accel3 = IMU3(index,6:8)';
    accel4 = IMU4(index,6:8)';
    accel5 = IMU5(index,6:8)';
    accelv = (accel + accel2 + accel3 + accel4 + accel5)/5;


    % predict states
    [quat, states, Tbn, delAng, delVel]  = ...
       PredictStates(quat,states,angRate,accel,dt);
    [quat2, states2, Tbn2, delAng2, delVel2]  = ...
       PredictStates(quat2,states2,angRate2,accel2,dt);
    [quat3, states3, Tbn3, delAng3, delVel3]  = ...
       PredictStates(quat3,states3,angRate3,accel3,dt);
    [quat4, states4, Tbn4, delAng4, delVel4]  = ...
       PredictStates(quat4,states4,angRate4,accel4,dt);
    [quat5, states5, Tbn5, delAng5, delVel5]  = ...
       PredictStates(quat5,states5,angRate5,accel5,dt);
    [quatv, statesv, Tbnv, delAngv, delVelv]  = ...
       PredictStates(quatv,statesv,angRatev,accelv,dt);

```

```
    % predict covariance matrix
    covariance  = ...
       PredictCovariance(delAng,delVel,quat,states,covariance,dt);
    covariance2  = ...
       PredictCovariance(delAng2,delVel2,quat2,states2,covariance2,dt);
    covariance3  = ...
       PredictCovariance(delAng3,delVel3,quat3,states3,covariance3,dt);
    covariance4  = ...
       PredictCovariance(delAng4,delVel4,quat4,states4,covariance4,dt);
    covariance5  = ...
       PredictCovariance(delAng5,delVel5,quat5,states5,covariance5,dt);
    covariancev  = ...
       PredictCovariance(delAngv,delVelv,quatv,statesv,covariancev,dt);


    measVelLog(:,index) = [0,0,0]';

    [quat,states,angErr,covariance,velInnov,velInnovVar] = ...
       FuseVelocity(quat,states,covariance,measVel);
    [quat2,states2,angErr2,covariance2,velInnov2,velInnovVar2] = ...
       FuseVelocity(quat2,states2,covariance2,measVel);
    [quat3,states3,angErr3,covariance3,velInnov3,velInnovVar3] = ...
       FuseVelocity(quat3,states3,covariance3,measVel);
    [quat4,states4,angErr4,covariance4,velInnov4,velInnovVar4] = ...
       FuseVelocity(quat4,states4,covariance4,measVel);
    [quat5,states5,angErr5,covariance5,velInnov5,velInnovVar5] = ...
       FuseVelocity(quat5,states5,covariance5,measVel);
    [quatv,statesv,angErrv,covariancev,velInnovv,velInnovVarv] = ...
       FuseVelocity(quatv,statesv,covariancev,measVel);

```

```
    velInnovLog(1,index) = time;
    velInnovLog(2:4,index) = velInnov;
    velInnovLog2(1,index) = time;
    velInnovLog2(2:4,index) = velInnov2;
    velInnovLog3(1,index) = time;
    velInnovLog3(2:4,index) = velInnov3;
    velInnovLog4(1,index) = time;
    velInnovLog4(2:4,index) = velInnov4;
    velInnovLog5(1,index) = time;
    velInnovLog5(2:4,index) = velInnov5;
    velInnovLogv(1,index) = time;
    velInnovLogv(2:4,index) = velInnovv;

    velInnovVarLog(1,index) = time;
    velInnovVarLog(2:4,index) = velInnovVar;
    velInnovVarLog2(1,index) = time;
    velInnovVarLog2(2:4,index) = velInnovVar2;
    velInnovVarLog3(1,index) = time;
    velInnovVarLog3(2:4,index) = velInnovVar3;
    velInnovVarLog4(1,index) = time;
    velInnovVarLog4(2:4,index) = velInnovVar4;
    velInnovVarLog5(1,index) = time;
    velInnovVarLog5(2:4,index) = velInnovVar5;
    velInnovVarLogv(1,index) = time;
    velInnovVarLogv(2:4,index) = velInnovVarv;


    angErrLog(1,index) = time;
    angErrLog(2,index) = angErr;
    angErrLog2(1,index) = time;
    angErrLog2(2,index) = angErr2;
    angErrLog3(1,index) = time;
```

```
    angErrLog3(2,index) = angErr3;
    angErrLog4(1,index) = time;
    angErrLog4(2,index) = angErr4;
    angErrLog5(1,index) = time;
    angErrLog5(2,index) = angErr5;
    angErrLogv(1,index) = time;
    angErrLogv(2,index) = angErrv;


    % read MAGnetometer measurements
    while ((MAG(MAGIndex,1) < IMU(index,1)) && (MAGIndex < MAGIndexlimit))
        MAGIndex = MAGIndex + 1;
        MAGBody = 0.001 * MAG(MAGIndex,3:5)';
        MAGBody2 = 0.001 * MAG2(MAGIndex,3:5)';
        MAGBody3 = 0.001 * MAG3(MAGIndex,3:5)';
        MAGBody4 = 0.001 * MAG4(MAGIndex,3:5)';
        MAGBody5 = 0.001 * MAG5(MAGIndex,3:5)';
        MAGBodyv = (MAGBody5+MAGBody4+MAGBody3+MAGBody2+MAGBody)/5;

        if (time ≥ 10 && headingAligned==0 && angErr < 1e-3)
            quat = AlignHeading(quat,MAGBody,measDec);
            quat2 = AlignHeading(quat2,MAGBody2,measDec);
            quat3 = AlignHeading(quat3,MAGBody3,measDec);
            quat4 = AlignHeading(quat4,MAGBody4,measDec);
            quat5 = AlignHeading(quat5,MAGBody5,measDec);
            quatv = AlignHeading(quatv,MAGBodyv,measDec);
            headingAligned = 1;
        end
        % fuse MAGnetometer measurements if new data available and when ...
           tilt has settled
        if (headingAligned == 1)
            [quat,states,covariance,decInnov,decInnovVar] = ...
```

```
                FuseMagnetometer(quat,states,covariance,MAGBody,measDec,Tbn);
            [quat2,states2,covariance2,decInnov2,decInnovVar2] = ...
                FuseMagnetometer(quat2,states2,covariance2,MAGBody2,measDec,Tbn2);
            [quat3,states3,covariance3,decInnov3,decInnovVar3] = ...
                FuseMagnetometer(quat3,states3,covariance3,MAGBody3,measDec,Tbn3);
            [quat4,states4,covariance4,decInnov4,decInnovVar4] = ...
                FuseMagnetometer(quat4,states4,covariance4,MAGBody4,measDec,Tbn4);
            [quat5,states5,covariance5,decInnov5,decInnovVar5] = ...
                FuseMagnetometer(quat5,states5,covariance5,MAGBody5,measDec,Tbn5);
            [quatv,statesv,covariancev,decInnovv,decInnovVarv] = ...
                FuseMagnetometer(quatv,statesv,covariancev,MAGBodyv,measDec,Tbnv);

            decInnovLog(1,MAGIndex) = time;
            decInnovLog(2,MAGIndex) = decInnov;
            decInnovVarLog(1,MAGIndex) = time;
            decInnovVarLog(2,MAGIndex) = decInnovVar;

            decInnovLog2(1,MAGIndex) = time;
            decInnovLog2(2,MAGIndex) = decInnov2;
            decInnovVarLog2(1,MAGIndex) = time;
            decInnovVarLog2(2,MAGIndex) = decInnovVar2;

            decInnovLog3(1,MAGIndex) = time;
            decInnovLog3(2,MAGIndex) = decInnov3;
            decInnovVarLog3(1,MAGIndex) = time;
            decInnovVarLog3(2,MAGIndex) = decInnovVar3;

            decInnovLog4(1,MAGIndex) = time;
            decInnovLog4(2,MAGIndex) = decInnov4;
            decInnovVarLog4(1,MAGIndex) = time;
            decInnovVarLog4(2,MAGIndex) = decInnovVar4;

```

```
            decInnovLog5(1,MAGIndex) = time;
            decInnovLog5(2,MAGIndex) = decInnov5;
            decInnovVarLog5(1,MAGIndex) = time;
            decInnovVarLog5(2,MAGIndex) = decInnovVar5;

            decInnovLogv(1,MAGIndex) = time;
            decInnovLogv(2,MAGIndex) = decInnovv;
            decInnovVarLogv(1,MAGIndex) = time;
            decInnovVarLogv(2,MAGIndex) = decInnovVarv;

        end

    end
    %% Master Fusion

        if MasterFusion == 1
            covariancem = inv(covariance(1:3,1:3)) + inv(covariance2...
                (1:3,1:3)) + inv(covariance3(1:3,1:3)) + inv(covariance4...
                (1:3,1:3)) + inv(covariance5(1:3,1:3));
            statesm = covariancem\( (covariance(1:3,1:3)\states(1:3)) + ...
                (covariance2(1:3,1:3)\states2(1:3)) + (covariance3...
                (1:3,1:3)\states3(1:3)) + (covariance4(1:3,1:3)\ ...
                states4(1:3)) + (covariance5(1:3,1:3)\states5(1:3)) );
        end

        if  MasterFusion == 0
            covariancev = (covariance(1:3,1:3)+covariance2(1:3,1:3)+ ...
                covariance3(1:3,1:3)+covariance4(1:3,1:3)+...
                covariance5(1:3,1:3) )/5;
            statesm = ...
               (states(2:4)+states2(2:4)+states3(2:4)+states4(2:4)...
                +states5(2:4))/5;
```

```
        end

        rotationMag = sqrt(statesm(1)^2 + statesm(2)^2 + statesm(3)^2);
        angErrLogm(index) = rotationMag;
        if rotationMag<1e-6
            ∆Quat = single([1;0;0;0]);
        else
            ∆Quat = [cos(0.5*rotationMag); ...
                [statesm(1);statesm(2);statesm(3)]/rotationMag*sin(0.5*rotationMag)];
        end

        % Update the quaternion states by rotating from the previous ...
            attitude through
        % the ∆ angle rotation quaternion
        quatm = [quatm(1)*∆Quat(1)-transpose(quatm(2:4))*∆Quat(2:4); ...
            quatm(1)*∆Quat(2:4) + ∆Quat(1)*quatm(2:4) + ...
            cross(quatm(2:4),∆Quat(2:4))];

        % normalise the updated quaternion states
        quatMag = sqrt(quatm(1)^2 + quatm(2)^2 + quatm(3)^2 + quatm(4)^2);
        if (quatMag > 1e-6)
            quatm = quatm / quatMag;
        end

        statesLogm(1,index) = time;
        statesLogm(2:4,index) = statesm;
        eulLogm(1,index) = time;
        quatLogm(:,index) = quatm;
        eulLogm(2:4,index) = QuatToEul(quatm);

        statesLog(1,index) = time;
        statesLog(2:10,index) = states;
```

```
        eulLog(1,index) = time;
        eulLog(2:4,index) = quat;
        quatLog(:,index) = quat;

        statesLog2(1,index) = time;
        statesLog2(2:10,index) = states2;
        eulLog2(1,index) = time;
        eulLog2(2:4,index) = QuatToEul(quat2);
        quatLog2(:,index) = quat2;

        statesLog3(1,index) = time;
        statesLog3(2:10,index) = states3;
        eulLog3(1,index) = time;
        eulLog3(2:4,index) = QuatToEul(quat3);
        quatLog3(:,index) = quat3;

        statesLog4(1,index) = time;
        statesLog4(2:10,index) = states4;
        eulLog4(1,index) = time;
        eulLog4(2:4,index) = QuatToEul(quat4);
        quatLog4(:,index) = quat4;

        statesLog5(1,index) = time;
        statesLog5(2:10,index) = states5;
        eulLog5(1,index) = time;
        eulLog5(2:4,index) = QuatToEul(quat5);
        quatLog5(:,index) = quat5;

        statesLogv(1,index) = time;
        statesLogv(2:10,index) = statesv;
        eulLogv(1,index) = time;
        eulLogv(2:4,index) = QuatToEul(quatv);
```

```
        quatLogv(:,index) = quatv;
end
```

```
%% Set Defaults
set(groot, 'DefaultTextInterpreter', 'LaTeX');
set(groot, 'DefaultAxesTickLabelInterpreter', 'LaTeX');
set(groot, 'DefaultAxesFontName', 'LaTeX');
set(groot, 'DefaultLegendInterpreter', 'LaTeX');
set(groot, 'DefaultAxesBox', 'on');
set(groot,'defaultLineLineWidth',1.5)
set(groot, 'defaultLegendLocation', 'best')
% set(groot, 'defaultFigureUnits','normalized')
% set(groot, 'defaultFigurePosition',[0 0 1 1]) % For full screen plots

%%
load('MAG_CALI_DATA.mat');

%% Mag1
mag1 = MAG(:,3:5);
[A1,c1] = MgnCalibration(mag1);
%% Mag2
mag2 = MAG2(:,3:5);
[A2,c2] = MgnCalibration(mag2);

%% Mag3
mag3 = MAG3(:,3:5);
[A3,c3] = MgnCalibration(mag3);

try
    plotcal == 1;
catch
```

```
    plotcal = 1;
    i = 1;
end



if plotcal == 1
    mag1 = (A1*(mag1'-repmat(c1,1,length(mag1))))';
    mag2 = (A2*(mag2'-repmat(c2,1,length(mag2))))';
    mag3 = (A3*(mag3'-repmat(c3,1,length(mag3))))';

    figure;
    subplot(1,2,1)
    plot3(mag1(:,1),mag1(:,2),mag1(:,3))
    legend('Calibrated')
    subplot(1,2,2)
    plot3(MAG(:,3),MAG(:,4),MAG(:,5))
    legend('UnCalibrated')

    figure;
    set(gca,'FontSize',20,'FontName','Times New Roman');
    subplot(1,2,1)
    plot3(mag2(:,1),mag2(:,2),mag2(:,3))
    legend('Calibrated')
    subplot(1,2,2)
    plot3(MAG2(:,3),MAG2(:,4),MAG2(:,5))
    legend('UnCalibrated')

    figure; set(gca,'FontSize',20,'FontName','Times New Roman');
    subplot(1,2,1)
    plot3(mag3(:,1),mag3(:,2),mag3(:,3))
    legend('Calibrated')
```

```
subplot(1,2,2)
plot3(MAG3(:,3),MAG3(:,4),MAG3(:,5))
legend('UnCalibrated')

%title('Magnetic Calibration')
xlabel('Mag X')
ylabel('Mag Y')
zlabel('Mag Z')

figure;
plot3(mag1(:,1),mag1(:,2),mag1(:,3),'.r');
hold on;
plot3(mag2(:,1),mag2(:,2),mag2(:,3),'.g');
plot3(mag3(:,1),mag3(:,2),mag3(:,3),'.b');
grid on; axis tight;
set(gca,'FontSize',20,'FontName','Times New Roman');
%title('Calibrated Magnetometer');
legend('IMU1','IMU2','IMU3','location','best','FontSize',20,'FontName','Times ...
    New Roman');
xlabel('MAGX');
ylabel('MAGY');
zlabel('MAGZ');

figure;
set(gca,'FontSize',20,'FontName','Times New Roman');
plot3(MAG(:,3),MAG(:,4),MAG(:,5),'.r');
hold on;
plot3(MAG2(:,3),MAG2(:,4),MAG2(:,5),'.g');
plot3(MAG3(:,3),MAG3(:,4),MAG3(:,5),'.b');
grid on; axis tight;
set(gca,'FontSize',20,'FontName','Times New Roman');
%title('UnCalibrated Magnetometer');
```

```
    legend('IMU1','IMU2','IMU3','location','best','FontSize',20,'FontName','Times ...
       New Roman');
    xlabel('MAGX');
    ylabel('MAGY');
    zlabel('MAGZ')


%      disp('Press Enter after checking MagCalibration:')
%      pause
end


% save('Calibration\CalVal.mat','A1','c1','A2','c2','A3','c3')
```

```
function [U,c] = MgnCalibration(X)

[N,m] = size(X);
if m>3&&N==3,X = X';N = m;m = 3;end;%check that X is not transposed
if N≤10,U = [];c = [];return;end;%not enough data no calibration !!
% write  the ellipsoid equation as D*p=0
% the best parameter is the solution of min||D*p|| with ||p||=1;
% form D matrix from X measurements
x = X(:,1); y = X(:,2); z = X(:,3);
D = [x.^2, y.^2, z.^2, x.*y, x.*z, y.*z, x, y, z, ones(N,1)];
D=triu(qr(D));%avoids to compute the svd of a large matrix
[U,S,V] = svd(D);%because usually N may be very large
p = V(:,end);if p(1)<0,p =-p;end;
% the following matrix A(p) must be positive definite
% The optimization done by svd does not include such a constraint
% With "good" data the constraint is allways satisfied
% With too poor data A may fail to be positive definite
```

```
% In this case the calibration fails
%
A = [p(1) p(4)/2 p(5)/2;
       p(4)/2 p(2) p(6)/2;
       p(5)/2 p(6)/2 p(3)];
[U,ok] = fchol(m,A);
if ¬ok,U = [];c = [];return;end%calibration fails too poor data!!
b = [p(7);p(8);p(9)];
v = Utsolve(U,b/2,m);
d = p(10);
s = 1/sqrt(v*v'-d);
c =-Usolve(U,v,m)';%ellipsoid center
U = s*U;%shape ellipsoid parameter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [A,ok] = fchol(n,A)
% performs Cholesky factoristation
A(1,1:n) = A(1,1:n)/sqrt(A(1,1));
A(2:n,1) = 0;
for j=2:n
  A(j,j:n) = A(j,j:n) - A(1:j-1,j)'*A(1:j-1,j:n);
  if A(j,j)≤0,ok=0;break;end%A is not positive definite
  A(j,j:n) = A(j,j:n)/sqrt(A(j,j));
  A(j+1:n,j) = 0;
end
ok=1;
function x=Utsolve(U,b,n)
x = zeros(1,length(n));
% solves U'*x=b
x(1) = b(1)/U(1,1);
for k=2:n
    x(k) = (b(k)-x(1:k-1)*U(1:k-1,k))/U(k,k);
end
```

```
function x=Usolve(U,b,n)
x = zeros(1,length(n));
% solves U*x=b
x(n) = b(n)/U(n,n);
for k=n-1:-1:1
    x(k) = (b(k)-U(k,k+1:n)*x(k+1:n)')/U(k,k);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

VITA

Ujjval Nirmalkumar Patel

Candidate for the Degree of

Master of Science

Thesis: SENSOR FUSION TO IMPROVE STATE ESTIMATE ACCURACY USING MULTIPLE INERTIAL MEASUREMENT UNITS AND IT'S APPLICATION TO WIND ESTIMATION

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University in December 2021.

Completed the requirements for Bachelors of Science in Aerospace Engineering at Oklahoma State University in May 2019.

Completed the requirements for Bachelors of Science in Mechanical Engineering at Oklahoma State University in May 2019.

Experience:

Graduate Research Assistant for Dr. Imraan Faruque at Applied Physics Group, Oklahoma State University from January 2020 to Present.

Professional Affiliations:

AIAA Student Member, APS Graduate Student Member, AMA Member.