RBE595 - Project 1b - Unscented Kalman Filter

Pranay Katyal
Worcester Polytechnic Institute
Worcester, MA, USA
pkatyal@wpi.edu

Anirudh Ramanathan
Worcester Polytechnic Institute
Worcester, MA, USA
aramanathan@wpi.edu

Hudson Kortus
Worcester Polytechnic Institute
Worcester, MA, USA
hkortus@wpi.edu

***Abstract*—This report for Project 1b seeks to showcase our understanding and implementation of the project 1b - Unscented Kalman Filter for Attitude Estimation. In the last assignment, we worked with IMU sensor and Vicon readings, and implemented attitude estimation with pure gyro, pure acceleration, by performing sensor fusion of the two using a Complementary Filter and Madgwick Filter. This project expands on complex implementation like using an Unscented Kalman Filter (UKF). We also plots the new data against the previous methods and true data from Vicon to showcase the comparison for all 6 of the datasets and test the implementation against 4 Test Datasets.**

Keywords - IMU, Gyro, Complimentary Filter, Vicon, Sensor Fusion, Madgwick Filter, Unscented Kalman Filter

I. Problem Statement

The objective of this project is to estimate the three-dimensional orientation (attitude) using acceleration and gyroscope measurements from a six-degree-of-freedom Inertial Measurement Unit (6-DoF IMU). Previously implemented methods to estimate orientation were performed by simply integrating the Gyroscope, integrating the Accelerometer, fusing these measurements integrations with a Complementary Filter, and fusing and filtering the accelerations with a madgwick filter. The objective of this project was to improve upon the previous methods by using an Unscented Kalman Filter alongside the sensor fusion using Quaternions.[1], [2], [3], [4], [5], [6], [7]

The Unscented Kalman Filter seeks to improve upon the Extended Kalman filter by truly accounting for nonlinear process models and system dynamics. Instead of creating a nonlinear process model, locally linearizing it with a Taylor Series, and operating on that linearized assumption, the Unscented Kalman Filter simply samples points on the state's distribution, projects them through the actual process noise, and measures how they changed. This allows the Unscented Kalman Filter to account for nonlinear processes and dynamics without ever having to model them. Therefore a well tuned Unscented Kalman Filter can perform faster and more accurately than an extended Kalman filter which must linearize its process model at every measurement.

II. Methodology

To start with the solution, we first need to get the sensor data into physical world values, Because the raw data is in incorrect format. The data for IMU was provided from imuRaw1.mat which had a corresponding ground truth data viconRot1.mat. both were stored in a dictionary format, and with a separate file containing timestamps ts.

A. Converting the Data

In imuRaw1, we had access to 'vals' and their corresponding 'ts'. Each column in 'vals' denotes data in the following order:

$$\begin{bmatrix} a_x & a_y & a_z & w_z & w_x & w_y \end{bmatrix}^T$$

Since these values are not in their physical units, we need to first convert them into their physical counterparts. We were provided access to their conversion formulas and we simply had to implement them.

To convert Acceleration values to ms^{-2} we used:

$$\tilde{a}_x = (a_x * s_x) + b_{a,x}$$

Here $\tilde{a}_x$ represents a_x in Physical units, and we do the same for a_y and a_z. $b_{a,x}$ is the Bias and s_x is the Scale factor.

We also have access to IMUParams.mat file which represents a 2 x 3 Matrix where the 1^{st} Row denotes the Scale factor values, and 2^{nd} row denotes the biases (computed as the average biases of all sequences using Vicon).

To convert Angular Velocity to $rad\ s^{-1}$ we used:

$$\tilde{w} = \frac{3300}{1023} \times \frac{\pi}{180} \times 0.3 \times (w - b_g)$$

Here $\tilde{w}$ represents the value of w in Physical units and b_g is the bias, which can be calculated as the average of the first few hundred samples (with the Assumption that the IMU is at rest at the beginning).

From viconRot1.mat we get access to 'rots' and the corresponding timestamps 'ts', Here rots represent a $3 \times 3 \times N$ matrix that denotes the ZXY Euler angles rotation matrix as estimated by the Vicon motion capture system.

B. Rig Used

Although we were not provided with a physical rig, we did have access to what it could have looked like. We can clearly see how the rig was setup, where the IMU is, and how the Vicon markers were attached to the setup.

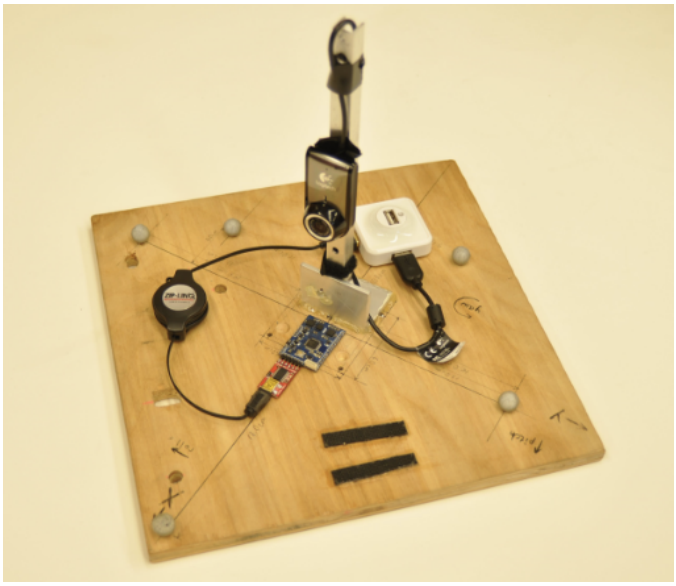

Fig. 1. Rig Used

Reflective Vicon markers are attached at known positions on the base, allowing the Vicon motion capture cameras to accurately track the 3D pose of the IMU during experiments. This arrangement ensured that the IMU orientation could be validated against the Vicon ground truth data while minimizing vibration and occlusion of the markers.

C. Sensor Calibration

One of the issues that we immediately encounter, is that the number of timestamps for both IMU data and Vicon data do not align, and they are also working for different amount of timestamps (although close but not exactly the same). the IMU data had 5645 timestamps, whereas Vicon data had 5561 timestamps, so we need to not only align them but also have them same amount of timestamps. This arises from the issue that IMU and Vicon are not Hardware synchronized, so we need to synchronize them in software.

D. SLERP and mask

We start with making the timestamps same!, we do this by simply creating a mask, that uses the Vicon 'ts' and then use it to modify IMU 'ts'. This mask essentially does a very basic filter, it puts the IMU 'ts' in the range with limits of vicon 'ts' and we save this as Valid 'ts'.

The best way to align these timestamps, and also get the correct vicon data for those timestamps is to use Slerp. It is called as Spherical Linear Interpolation, and is a way to help us sample the correct Vicon data for the Valid 'ts'. We do this by first creating a Slerp object using the Scipy library's inbuilt function, that takes in the Vicon 'ts' and Vicon 'rots' in matrix form and then we use IMU 'ts', more specifically the Valid IMU 'ts' to sample the Valid vicon values from the Slerp object.

III. Implementation

A. Calculating Orientation from Gyro only

In this section we evaluate orientation from the IMU's Gyro readings, which provide us the Angular Velocity w. For this we utilized a numerical integration method known as dead reckoning. We Assumed that the initial orientation for IMU is same as Vicon since we need a starting angle for numerical integration to work.

The gyroscope provides angular velocity measurements $\tilde{w}$ along the three axes. To estimate orientation, we integrate the angular velocity measurements over time. We initialize the Euler angles with the first Vicon reading as ground truth, and then update them at each timestep using numerical integration. The incremental update is expressed as

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_{t+1} = \begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}_{t} + \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}_{t} \delta t$$

where ϕ, θ, and ψ represent roll, pitch, and yaw, respectively, and $\dot{\phi}, \dot{\theta}, \dot{\psi}$ are the corresponding angular velocity components from the gyroscope. This simple dead-reckoning approach captures short-term orientation changes, but because biases and noise accumulate over time, the estimates drift significantly without external correction.

Although this method captures fast dynamics, it suffers from drift due to the accumulation of bias and noise in the gyroscope. Without correction from another sensor, the orientation estimate deviates significantly from ground truth over time.

B. Calculating Orientation from Acceleration only

The accelerometer provides measurements of both gravity and linear accelerations. When the IMU is relatively stationary or moving slowly, the accelerometer can be used to estimate the tilt (pitch and roll) by assuming that the measured acceleration corresponds primarily to gravity. The tilt angles are computed as:

$$\theta = \arctan 2\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right), \quad \phi = \arctan 2\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

where θ and ϕ denote pitch and roll respectively. Yaw cannot be reliably recovered from the accelerometer alone, since gravity provides no information about rotation around the vertical axis, considering the fact that we assumed IMU is vertical. We do have formulation for it, in case IMU is not vertically aligned, but here we assumed it is for simplicity.

This method provides stable long-term estimates without drift, but is highly sensitive to dynamic motion. Rapid translational accelerations corrupt the gravity estimate, leading to noisy and unreliable attitude estimation during fast maneuvers.

C. Calculating Orientation from Sensor Fusion using Complimentary Filter

To leverage the strengths of both sensors, we implemented a complementary filter. The idea is to use the gyroscope for short-term orientation updates, while correcting long-term drift using accelerometer-based tilt estimates. The complementary filter blends the two sources as:

$$\hat{R}_t = \alpha\, \hat{R}_{acc,t} + (1-\alpha)\, \hat{R}_{gyro,t}$$

where $\hat{R}_{gyro,t}$ is the orientation propagated from gyroscope integration, $\hat{R}_{acc,t}$ is the orientation estimated from accelerometer tilt, and $\alpha \in [0,1]$ is a tuning parameter controlling the balance between fast response and drift correction. [1]

In practice, α was tuned experimentally (e.g., $\alpha = 0.9995$) to allow the gyroscope to dominate high-frequency dynamics, while the accelerometer provided a low-frequency correction. This significantly improved attitude estimation, especially for pitch and roll. Yaw remained dependent on the gyroscope due to the lack of absolute heading information from the accelerometer or Vicon system.

IV. Calculating Orientation from Sensor Fusion using Madgwick Filter

While the Complementary Filter fuses gyroscope and accelerometer information in Euler space, it has two main limitations:

- the accelerometer signal is corrupted by translational accelerations, which reduces reliability of long-term correction
- the use of Euler angles introduces singularities (gimbal lock)

To overcome these issues, we implemented the Madgwick Filter, which works in quaternion space and performs a gradient-descent correction step based on accelerometer data. This avoids singularities and improves long-term stability.

A. Initialization

The filter starts with an initial orientation. For training datasets, this is taken from the first Vicon reading, converted to a quaternion using For test datasets without Vicon, the initial orientation is assumed to be zero (identity quaternion). We use the formula below to get the initial quaternion. `R.from_euler('zxy', initial_orientation, degrees=True).as_quat(scalar_first=True)`

B. Gyroscope Update

At each time step, the body angular velocity $\omega = [\omega_x, \omega_y, \omega_z]^T$ is first expressed as a quaternion

$$\omega_q = \begin{bmatrix} 0 & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

and the quaternion derivative from the gyroscope is computed as

$$\dot{q}_\omega = \tfrac{1}{2}\, q \otimes \omega_q$$

where $\otimes$ denotes quaternion multiplication. This corresponds to dead-reckoning integration of gyroscope data.

$$q_1 \otimes q_2 = \begin{bmatrix} w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2 \\ w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2 \\ w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2 \\ w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2 \end{bmatrix}$$

C. Accelerometer Correction

To prevent drift, a correction step is applied using normalized accelerometer measurements $\hat{a} = [a_x, a_y, a_z]^T / \|a\|$.

The objective is to minimize the error between the expected gravity vector from the quaternion and the measured gravity vector from the accelerometer. When the IMU is stationary or moving slowly, the accelerometer primarily measures gravity, which should appear as $[0,0,1]^T$ in the global reference frame. Using the current quaternion estimate $q = [w,x,y,z]$, we can predict what this gravity vector should look like in the sensor frame by rotating it through the inverse quaternion transformation.

The residual function f captures the mismatch between this predicted gravity direction and the actual accelerometer measurement:

$$f = \begin{bmatrix} 2(xz - wy) - a_x \\ 2(wx + yz) - a_y \\ 2(0.5 - x^2 - y^2) - a_z \end{bmatrix}$$

Each component represents the error along one sensor axis. When $f = 0$, the quaternion estimate perfectly aligns with the measured gravity direction.

To minimize this error using gradient descent, we need the Jacobian matrix J, which describes how small changes in each quaternion component affect the residual error:

$$J = \begin{bmatrix} -2y & 2z & -2w & 2x \\ 2x & 2w & 2z & 2y \\ 0 & -4x & -4y & 0 \end{bmatrix}$$

Each column of J represents the partial derivatives of f with respect to one quaternion component (w, x, y, z respectively). This matrix encodes the sensitivity of the gravity prediction error to changes in orientation.

The gradient of the error function is computed as:

$$\nabla f = J^T f$$

This gradient points in the direction of steepest increase in error. To reduce the error, we move in the opposite direction, scaled by a tuning parameter β:

$$\dot{q}_\nabla = -\beta\, \frac{\nabla f}{\|\nabla f\|}$$

The normalization ensures that the correction magnitude is independent of the error scale, providing stable convergence behavior. The parameter β controls how aggressively the filter corrects for accelerometer-based errors versus trusting the gyroscope integration.

D. Quaternion Update

The Madgwick filter combines information from both sensors by fusing the gyroscope-based quaternion derivative with the accelerometer-based correction term. The total quaternion derivative represents the complete rate of change in orientation:

$$\dot{q} = \dot{q}_\omega + \dot{q}_\nabla$$

where $\dot{q}_\omega$ captures the rotational dynamics from the gyroscope, and $\dot{q}_\nabla$ provides the gravity-based correction from the accelerometer.

This combined rate of change is integrated forward in time using Euler's method:

$$q_{t+1} = q_t + \dot{q}\,\Delta t$$

where Δt is the time step between consecutive IMU measurements. This integration step propagates the orientation estimate from the previous time step to the current one, incorporating both the predicted motion and the measured correction.

However, numerical integration can introduce small errors that cause the quaternion to deviate from unit length. Since valid rotation quaternions must have unit magnitude, normalization is essential:

$$q_{t+1} = \frac{q_{t+1}}{\|q_{t+1}\|}$$

This normalization step ensures that the quaternion remains on the unit sphere in 4D space, preserving its validity as a rotation representation. Without this step, accumulated numerical errors would eventually cause the quaternion to represent invalid transformations, leading to incorrect orientation estimates.

The normalized quaternion q_{t+1} now serves as the updated orientation estimate for the current time step, ready to be converted to Euler angles for visualization or used as the starting point for the next iteration of the filter.

V. Calculating Orientation using Unscented Kalman Filter

The Unscented Kalman Filter (UKF) differs from simpler attitude estimation methods, such as the Madgwick and Complementary filters, in handling uncertainty and nonlinearity. The UKF propagates a set of sigma points through the nonlinear process and measurement models, updating both the state and covariance using Kalman gain, which provides statistically consistent estimates under Gaussian noise assumptions. In contrast, the Madgwick filter applies a gradient-descent optimization to directly minimize the error between measured and predicted sensor outputs, and the Complementary filter blends gyroscope integration with accelerometer/magnetometer corrections using a fixed weighting. While these methods are computationally efficient, they do not maintain an explicit representation of state uncertainty. The UKF, however, is built upon a full probabilistic framework, where the careful construction and propagation of sigma points is central to the algorithm.

A. Sigma Point Generation

The full state vector is represented in 7 dimensions:

$$x = \begin{bmatrix} q \\ \omega \end{bmatrix}, \quad q \in \mathbb{R}^4,\ \omega \in \mathbb{R}^3$$

where q is the unit quaternion describing attitude, and ω is the angular velocity.

However, the quaternion is subject to the normalization constraint:

$$\|q\| = 1$$

This constraint reduces the degrees of freedom of the quaternion from 4 to 3, since each parameter is dependent on one another. As a result, the effective dimension of the state is 6D.

For this reason, the UKF constructs sigma points in the reduced 6D space, which are then mapped back into the 7D representation for propagation.

The UKF represents the state distribution not by a single mean and covariance, but by a set of carefully chosen *sigma points*. These sigma points capture the mean and covariance of the state distribution and can be propagated through the nonlinear process and measurement models to approximate their effects on the distribution.

$x_{\text{mean}} \in \mathbb{R}^7$: current state estimate (quaternion + angular velocity)

$P \in \mathbb{R}^{6\times 6}$: current covariance matrix in reduced space

$Q \in \mathbb{R}^{6\times 6}$: process noise covariance

The process noise is incorporated into the state covariance:

$$P_{\text{aug}} = P + Q$$

The square root of the covariance is computed using the Cholesky factorization (with regularization and SVD fallback for numerical robustness). This yields a decomposition:

$$P_{\text{aug}} = SS^T$$

The result is scaled according to:

$$S \leftarrow S\sqrt{2n}$$

where $n = 6$ is the dimension of the reduced state space. Thus, $2n = 12$ sigma points are generated.

The sigma points in reduced space are defined as the positive and negative columns of S:

$$\sigma_i^{(6)} = +S_{:,i}, \quad \sigma_{i+n}^{(6)} = -S_{:,i}, \quad i = 1, \ldots, n$$

Each 6D perturbation vector is split into:

$$\sigma^{(6)} = \begin{bmatrix} \delta r \\ \delta\omega \end{bmatrix}$$

where $\delta r \in \mathbb{R}^3$ is a small rotation vector and $\delta\omega \in \mathbb{R}^3$ is an angular velocity perturbation.

The rotation vector δr is converted into a quaternion noise term and the quaternion mean q_{mean} is updated:

$$q_{\text{new}} = || \left(q_{\text{mean}} \otimes q_{\delta} \right) ||$$

The angular velocity mean is updated additively:

$$\omega_{\text{new}} = \omega_{\text{mean}} + \delta\omega$$

Finally, the perturbed sigma point in 7D state space is constructed:

$$x^{(7)} = \begin{bmatrix} q_{\text{new}} \\ \omega_{\text{new}} \end{bmatrix}$$

The complete set of sigma points is:

$$\mathcal{X} = \{x_1^{(7)}, x_2^{(7)}, \ldots, x_{2n}^{(7)}\}, \quad \mathcal{X} \in \mathbb{R}^{2n \times 7}$$

The sigma points allow the UKF to approximate the nonlinear transformation of a Gaussian distribution without requiring linearization.

B. UKF Prediction

The process model describes how the system state evolves over time, independent of sensor measurements. It models the natural dynamics of orientation based on angular velocity.

The prediction step propagates each sigma point through the process model using quaternion integration. For each sigma point $\chi_i = [q_i^T, \omega_i^T]^T$, the process model applies the following transformations:

The angular velocity remains constant over the prediction interval:

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

For the quaternion update, the total rotation angle over the time step Δt is computed as:

$$\alpha = \|\omega_k\| \, \Delta t$$

When $\alpha > 10^{-8}$, the normalized rotation axis is:

$$\hat{\omega} = \frac{\omega_k}{\|\omega_k\|}$$

The quaternion increment representing this rotation is constructed as:

$$q_{\Delta} = \begin{bmatrix} \cos(\alpha/2) \\ \hat{\omega}_x \sin(\alpha/2) \\ \hat{\omega}_y \sin(\alpha/2) \\ \hat{\omega}_z \sin(\alpha/2) \end{bmatrix}$$

For very small rotations ($\alpha \leq 10^{-8}$), the identity quaternion is used: $q_{\Delta} = [1, 0, 0, 0]^T$.

The updated orientation is obtained through quaternion multiplication:

$$q_{k+1} = \text{normalize}(q_k \otimes q_{\Delta})$$

The predicted sigma point becomes:

$$\chi_i^{k+1} = \begin{bmatrix} q_{k+1} \\ \omega_{k+1} \end{bmatrix}$$

This process model captures the natural integration of angular velocity into orientation changes while maintaining the quaternion unit norm constraint. The transformation is applied to all 12 sigma points, generating the predicted sigma point set for subsequent mean and covariance computation.

C. Computing Quaternion Mean

Taking the mean of the prediction $\mathcal{Y}_i$ is more complex than the simple expectation $\mathbb{E}$ because the sigma points represent orientations which are periotic (-179 is one less than 179). To find the true mean of $\mathcal{Y}_i$ we must run intrinsic gradient descent to find the mean.

Algorithm 1 Quaternion Mean Update

Data: $\{q_1, q_2, \ldots, q_{2n}\}$
Result: $\bar{q}$, error vectors
Initialize $\bar{q}$ as q_1
while $t < \text{MaxIter}$ **and** $\|\vec{e}\| > \text{Threshold}$ **do**
  for all $i = 1, \ldots, 2n$ **do**
    $q_{rel,i} \leftarrow q_i \otimes \bar{q}^*$ ▷ $\bar{q}^*$ is quaternion conjugate
    $\vec{e}_i \leftarrow \text{QuaternionToRotVec}(q_{rel,i})$
  end for
  $\vec{e} \leftarrow \frac{1}{2n} \sum_{i=1}^{2n} \vec{e}_i$ ▷ Mean error vector
  $q_{adj} \leftarrow \text{RotVecToQuaternion}(\vec{e})$ ▷ Convert to quaternion
  $\bar{q} \leftarrow q_{adj} \otimes \bar{q}$ ▷ Apply adjustment
  $\bar{q} \leftarrow \bar{q} / \|\bar{q}\|$ ▷ Normalize
end while

In this algorithm a random $\bar{q}$ is initialized (in our case off of the first element in the sigma points for simplicity) and its inverse is multiplied with each quaternion in the prediction (row of $\mathcal{Y}_i$) to create an error vector e_i. That error vector is then averaged with all other error vectors to create the barycentric mean $\vec{(e)}$ called the adjustment vector. The adjustment vector is a rotation vector that points in the direction of the real mean. $\vec{e_i}$ vector is converted to a quaternion representation e and multiplied against the current guess $\bar{q}_t$ to get the next guess $\bar{q}_{t+1}$. This algorithm is repeated until we run a predetermined number of iterations or $\vec{e}$ goes below a threshold.

D. Computing 6D Covariance

Computing the prediction covariance involves transforming the predicted sigma points from their 7D representation back to the 6D uncertainty space used for covariance propagation. This transformation is necessary because the quaternion constraint reduces the effective DoF from 7 to 6.

The predicted sigma points $\mathcal{Y}_i \in \mathbb{R}^7$ contain both quaternion and angular velocity components. To compute the covariance in the reduced space, each sigma point must be converted to a 6D deviation vector $\mathcal{W}_i \in \mathbb{R}^6$:

$$\mathcal{W}_i = \begin{bmatrix} \vec{r}_i \\ \Delta\omega_i \end{bmatrix}$$

where: $\vec{r}_i \in \mathbb{R}^3$ is the rotation vector obtained from the quaternion mean computation (the error vector from the iterative algorithm) $\Delta\omega_i = \omega_i - \bar{\omega} \in \mathbb{R}^3$ is the angular velocity deviation from the computed mean $\bar{\omega}$

The quaternion components are converted to rotation vectors through the iterative mean algorithm, which naturally produces

the error vectors $\vec{r}_i$ representing the rotational deviation of each sigma point from the quaternion mean.

Once all sigma points are transformed to the 6D representation, the prediction covariance matrix is computed as:

$$P_{k|k-1}^{(6)} = \frac{1}{2n}\sum_{i=1}^{2n}\mathcal{W}_i\mathcal{W}_i^T = \frac{1}{2n}\mathcal{W}^T\mathcal{W}$$

where $\mathcal{W} \in \mathbb{R}^{12\times 6}$ is the matrix containing all deviation vectors as rows, and $2n = 12$ is the total number of sigma points.

Note that this process reverses the matrix square root computation performed during sigma point generation. While sigma point generation uses Cholesky decomposition when the covariance matrix is well-conditioned, it falls back to SVD decomposition for numerical robustness when Cholesky factorization fails due to ill-conditioning.

This covariance matrix captures the uncertainty propagation through the nonlinear process model while respecting the quaternion constraint, and serves as input for subsequent measurement update steps.

E. Sigma Point Transformation through Gyroscope Model

Because the gyroscope directly measures angular rotation, the measurement model is simply the raw gyro measurement predictions $\vec{\omega_k}$ plus the process noise $\vec{v_{rot}}$.

$$z_k = \vec{\omega_k} + \vec{v_{rot}}$$

In practice, we withhold adding the measurement noise until we compute our Kalman gain to keep the model more stable.

The implementation accounts for coordinate frame conventions and sensor mounting orientation through careful axis management. The IMU provides measurements in a ZXY convention, which must be reordered to match the filter's XYZ coordinate frame. This reordering operation, implemented as `omega[[1,2,0], i]`, transforms the raw sensor data into the expected measurement space.

The choice of ZXY Euler angle convention for the final orientation output aligns with the Vicon ground truth data format, enabling direct comparison and validation. This convention affects both the initial state setup, where Vicon orientations are converted using `R.from_euler('zxy', degrees=True)`, and the final output conversion from quaternions back to Euler angles. The measurement model maintains consistency with this coordinate frame choice throughout the estimation process.

F. Sigma Point Transformation through Accelerometer Model

The measurement model is another function that describes the relationship between states and sensor readings, independent of the measurement from inputs. It is usually modeled after the data provided by sensors mounted on the system.

The selected sigma points are passed through the accelerometer measurement model to generate a corresponding set of predicted measurements. The purpose of this transformation is to relate the state estimate from the process model to the expected accelerometer outputs.

Each predicted sigma point contains a quaternion component that encodes the body's orientation. This quaternion is first normalized to preserve unit length. The global gravity vector is defined in the world frame as

$$g = \begin{bmatrix} 0 \\ 0 \\ 9.81 \end{bmatrix}$$

The accelerometer model transforms this vector into the body frame using quaternion rotation. A three-dimensional vector is first expressed as a *vector quaternion*:

$$g = (0, g_x, g_y, g_z)$$

and then rotated using the quaternion q associated with the sigma point:

$$g' = q\, g\, q^{-1}$$

where q^-1 is the quaternion conjugate. This gives the gravity vector expressed in the body frame.

The accelerometer measurement model is then written as:

$$z_{\text{acc}} = g' + f_{\text{acc}}$$

where f_{acc} represents accelerometer measurement noise. The vector part of

$$z_{\text{acc}} = \begin{bmatrix} g'_x \\ g'_y \\ g'_z \end{bmatrix}$$

is extracted as the predicted accelerometer measurement for that sigma point. Repeating this transformation for all sigma points yields a set of predicted measurement sigma points, which are later used to compute the expected mean measurement, covariance, and cross-covariance with the state.

G. Computing Cross-Covariance

The cross-covariance matrix P_{xz} quantifies the correlation between state estimation errors and measurement prediction errors. This matrix is essential for computing the Kalman gain and determines how measurement innovations should update the state estimate.

Given the 6D state deviation matrix $\mathcal{W} \in \mathbb{R}^{12\times 6}$ from the covariance computation and the measurement deviations $\mathcal{Z}_{dev} \in \mathbb{R}^{12\times m}$ (where $m = 3$ for accelerometer or gyroscope measurements), the cross-covariance is computed as:

$$P_{xz} = \frac{1}{2n}\mathcal{W}^T\mathcal{Z}_{dev}$$

where $2n = 12$ is the number of sigma points. Each row of $\mathcal{W}$ contains the 6D deviation vector for one sigma point:

$$\mathcal{W}_i = \begin{bmatrix} \vec{r}_i \\ \Delta\omega_i \end{bmatrix}$$

where $\vec{r}_i$ is the rotation vector from the quaternion mean computation and $\Delta\omega_i = \omega_i - \bar{\omega}$ is the angular velocity deviation.

The measurement deviations are computed as:

$$\mathcal{Z}_{dev,i} = z_i - \bar{z}$$

where z_i is the predicted measurement from sigma point i and $\bar{z}$ is the mean predicted measurement.

This cross-covariance captures how uncertainties in the state (orientation and angular velocity) correlate with uncertainties in the measurements, enabling optimal fusion of sensor information with the predicted state.

H. Computing Kalman Gain

The Kalman gain determines how much the filter should correct the predicted state estimate based on incoming measurements. The function implements this computation using the state–measurement covariance terms.

The required matrices are:

$$P_{xz} \in \mathbb{R}^{n\times m}$$

$$P_{zz} \in \mathbb{R}^{m\times m}$$

$$R \in \mathbb{R}^{m\times m}$$

Here, $n = 6$ (state dimension in the reduced representation) and $m = 3$ (measurement dimension).

The total uncertainty in the measurement is given by the innovation covariance:

$$P_{vv} = P_{zz} + R$$

This term accounts for both the predicted measurement uncertainty and the noise characteristics of the sensor. It quantifies how reliable the innovation (the difference between actual and predicted measurements) is expected to be.

The Kalman gain is then computed as:

$$K = P_{xz} P_{vv}^{-1}$$

The Kalman gain balances the trust between the process model and the measurement:

$$R \text{ small } \Rightarrow \ K \text{ large } \Rightarrow \text{ trust the measurement more}$$

$$R \text{ large } \Rightarrow \ K \text{ small } \Rightarrow \text{ trust the process model more}$$

Once computed, the Kalman gain is used to update the state estimate:

$$x_k = \hat{x}_{k|k-1} + K\left(z_k - \hat{z}_k\right)$$

Here:

$$\hat{x}_{k|k-1} \text{ : predicted state estimate}$$

$$z_k \text{ : actual measurement}$$

$$\hat{z}_k \text{ : predicted measurement from the measurement model}$$

I. Sequential Measurement Updates

The UKF implementation processes gyroscope and accelerometer measurements sequentially rather than simultaneously. This approach offers computational advantages and improved numerical stability compared to joint measurement processing.

First, the predicted sigma points are transformed through the gyroscope measurement model, and the resulting innovation is used to update both the state estimate and covariance matrix. The updated state then serves as the starting point for the accelerometer measurement update. This sequential processing allows each sensor's characteristics to be handled independently while maintaining optimal fusion properties.

The gyroscope measurements require axis reordering from the IMU's native ZXY convention to the standard XYZ convention used in the filter's coordinate frame. This transformation accounts for the physical mounting orientation of the IMU sensor and ensures consistency with the quaternion-based state representation. The measurement model directly relates the angular velocity state components to the reordered gyroscope readings, making this sensor update computationally efficient since no nonlinear transformations are required.

J. Updating UKF Values

The update step fuses the actual sensor measurements with the predicted state in order to correct the estimate.

Innovation: The innovation is the difference between the actual measurement and the predicted measurement:

$$\nu = z_{\text{actual}} - z_{\text{predicted}}$$

State Update in Reduced Space: The Kalman gain is applied to the innovation to compute the state correction in the reduced 6D space:

$$\Delta x^{(6)} = K\,\nu$$

This update vector is partitioned as:

$$\Delta x^{(6)} = \begin{bmatrix} \Delta r \\ \Delta\omega \end{bmatrix}$$

where $\Delta r \in \mathbb{R}^3$ represents a small rotation update and $\Delta\omega \in \mathbb{R}^3$ is the angular velocity update.

The small rotation vector Δr is converted into a correction quaternion and an updated quaternion is obtained:

$$q_{\text{updated}} = ||\left(q_\Delta \otimes q_{\text{mean}}\right)||$$

The angular velocity is updated additively:

$$\omega_{\text{updated}} = \omega_{\text{mean}} + \Delta\omega$$

The corrected state estimate is assembled as:

$$x^{(7)}_{\text{updated}} = \begin{bmatrix} q_{\text{updated}} \\ \omega_{\text{updated}} \end{bmatrix}$$

Finally, the covariance matrix is updated to reflect the reduced uncertainty after incorporating the measurement:

$$P^{(6)}_{\text{updated}} = P^{(6)} - K\,\left(P_{zz} + R\right)K^T$$

K. Orientation from UKF

The complete UKF implementation for orientation estimation consists of the following algorithmic steps, executed at each time step k:

Initialization:

- Initialize state vector: $x_0 = [q_0^T, \omega_0^T]^T \in \mathbb{R}^7$
- Initialize 6D covariance matrix: $P_0 \in \mathbb{R}^{6\times6}$
- Set process noise matrix: $Q \in \mathbb{R}^{6\times6}$
- Set measurement noise matrices: $R_{gyro}, R_{accel} \in \mathbb{R}^{3\times3}$

Main Filter Loop:

Step 1: Sigma Point Generation

- Compute augmented covariance: $P_{aug} = P_{k-1} + Q$
- Extract matrix square root using Cholesky decomposition (with SVD fallback)
- Generate 12 sigma points in 6D space, then map to 7D state space

Step 2: Prediction Phase

- Propagate each sigma point through process model using quaternion integration
- Compute quaternion mean using iterative gradient descent algorithm
- Compute angular velocity mean using standard averaging
- Assemble predicted state: $\hat{x}_{k|k-1} = [\hat{q}_{k|k-1}^T, \hat{\omega}_{k|k-1}^T]^T$
- Transform sigma points to 6D deviation space and compute predicted covariance: $P_{k|k-1}$

Step 3: Gyroscope Measurement Update

- Transform predicted sigma points through gyroscope measurement model
- Compute predicted gyroscope measurement mean and covariance
- Calculate cross-covariance between state and gyroscope measurements
- Compute Kalman gain: $K_{gyro} = P_{xz}(P_{zz} + R_{gyro})^{-1}$
- Update state and covariance using gyroscope innovation

Step 4: Accelerometer Measurement Update

- Transform predicted sigma points through accelerometer measurement model
- Rotate gravity vector using quaternion rotation: $g_{body} = q \otimes g_{global} \otimes q^*$
- Compute predicted accelerometer measurement mean and covariance
- Calculate cross-covariance between state and accelerometer measurements
- Compute Kalman gain: $K_{accel} = P_{xz}(P_{zz} + R_{accel})^{-1}$
- Update state and covariance using accelerometer innovation

Step 5: Output Generation

- Normalize final quaternion: $q_k = q_k / \|q_k\|$
- Convert quaternion to ZXY Euler angles for comparison with ground truth
- Store orientation estimate for visualization and analysis

This algorithmic framework successfully fuses gyroscope and accelerometer measurements while maintaining the nonlinear quaternion constraint, providing superior attitude estimation compared to other filtering approaches.

VI. Results

We successfully carried out the project, major learning curve for us, was getting to learn about the Unscented Kalman Filters and how to implement it. But as a result of this project, we now have learnt about UKF, and how to utilize them to get way better estimates of attitude as compared to other methods. We also learnt about how to apply quaternion and use them alongside UKF.

The results for complimentary filter now perform better at values like $\alpha = 0.99$ because we were able to fix our acceleration value issues from project 0. This also further improved the Madgwick filter.

The p1a implementation effectively balances the short-term accuracy of the gyroscope with the long-term stability of the accelerometer. The tuning parameter β (set to 0.01 in our experiments) controls the correction strength. With proper tuning, the Madgwick filter tracked the Vicon ground truth more accurately than the gyroscope-only and Complementary Filter approaches, especially for roll and pitch, while avoiding the drift observed in yaw. Even though the recommended value by the Author of the Madgwick filter is 0.041, upon testing around multiple values like $0.0, 0.8, 0.2, 0.1$, we found that 0.01 was a clear winner for us.[8]

For the p1b's UKF implementation, we ended up with finding Q and R values, that maximized performance on each of the 6 Training datasets. While they work decent on other datasets, they outperform each other on their own relevant dataset.

TABLE I
TUNING PARAMETERS FOR DIFFERENT DATASETS

Dataset	$Q = \text{diag}(\cdot)$	$R = \text{diag}(\cdot)$
1	[10, 10, 10, 0.5, 0.5, 0.5]	R_{gyro}: [0.05, 0.05, 0.05]; R_{accel}: [5.0, 5.0, 5.0]
2	[9, 9, 9, 0.5, 0.5, 0.5]	R_{gyro}: [0.05, 0.05, 0.05]; R_{accel}: [6.0, 6.0, 6.0]
3	[1, 1, 1, 0.0005, 0.0005, 0.0005]	R_{gyro}: [0.05, 0.05, 0.05]; R_{accel}: [1000, 1000, 1000]
4	[10, 10, 10, 9, 9, 9]	R_{gyro}: [0.1, 0.1, 0.1]; R_{accel}: [0.001, 0.001, 0.001]
5	[15, 10, 10, 0.15, 0.15, 0.15]	R_{gyro}: [1, 1, 1]; R_{accel}: [350, 350, 350]
6	[32, 32, 32, 0.15, 0.15, 15]	R_{gyro}: [0.005, 0.005, 0.005]; R_{accel}: [900, 900, 900]
7	[3.4, 3.4, 3.4, 0.5, 0.5, 0.5]	R_{gyro}: [15, 15, 15]; R_{accel}: [15, 15, 25]
8	[3.4, 3.4, 3.4, 0.5, 0.5, 0.5]	R_{gyro}: [15, 15, 15]; R_{accel}: [15, 15, 25]
9	[3.4, 3.4, 3.4, 0.5, 0.5, 0.5]	R_{gyro}: [15, 15, 15]; R_{accel}: [15, 15, 25]
10	[3.4, 3.4, 3.4, 0.5, 0.5, 0.5]	R_{gyro}: [15, 15, 15]; R_{accel}: [15, 15, 25]

The UKF implementation revealed significant dataset-dependent tuning requirements, as shown in Table (I) . The process noise matrix Q exhibits substantial variation across datasets, with orientation uncertainty ranging from 1 to 32 and angular velocity uncertainty spanning from 0.0005 to 15. The measurement noise parameters show even greater disparity, particularly for accelerometer noise R_{accel}, which varies across three orders of magnitude from 0.001 to 1000. This wide parameter variation indicates fundamentally differ-

ent motion characteristics across the training datasets. Datasets 3 and 6 require extremely high accelerometer noise values (1000 and 900 respectively), effectively reducing the filter to gyroscope-dominated operation during periods of high external acceleration or sensor disturbance. Conversely, Dataset 4 demands aggressive accelerometer corrections (R_{accel} = 0.001) coupled with high angular velocity process noise, suggesting scenarios where accelerometer information is highly reliable but angular motion is unpredictable.

On the Test Dataset, since we needed to Fix up on a singular tune rather than over-tuning the UKF each time for each dataset, we were experimentally able to find a tune that Generally works for All of the Training and Test Datasets which would be :

$$Q = \begin{bmatrix} 3.4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

$$R_{\text{gyro}} = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 15 \end{bmatrix} \quad R_{\text{accel}} = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

While these don't perform "best" always, they outperform Madgwick filter for a good set of times.

A. *Orientation Plots*

We also plotted $Z - X - Y$ Euler angles from each of the three implementations against the Vicon's data. which is shown below :

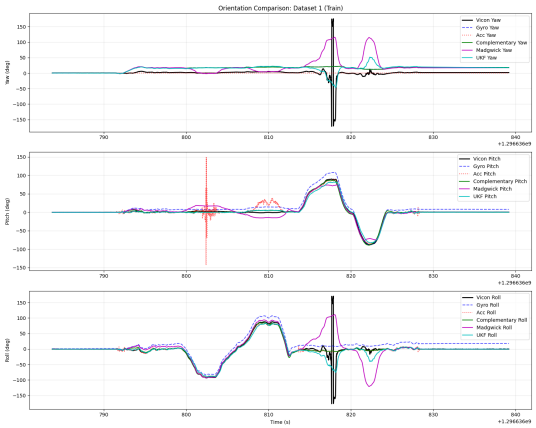

Fig. 2. Dataset 1

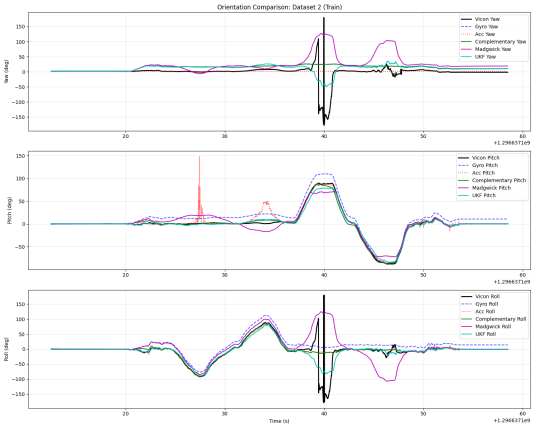

Fig. 3. Dataset 2

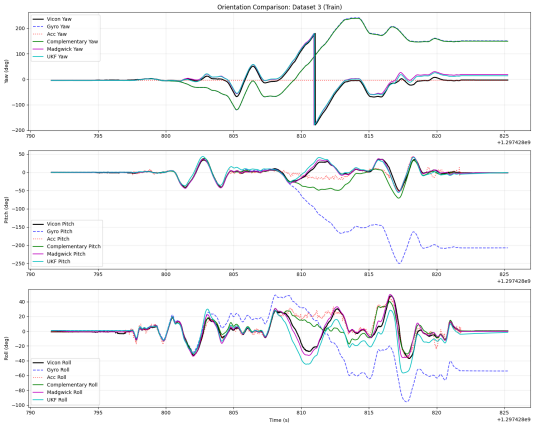

Fig. 4. Dataset 3

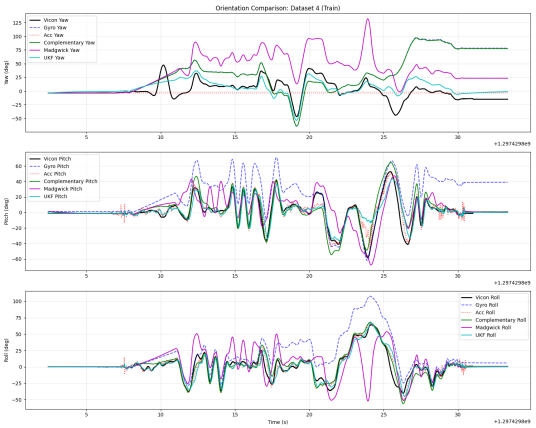

Fig. 5. Dataset 4

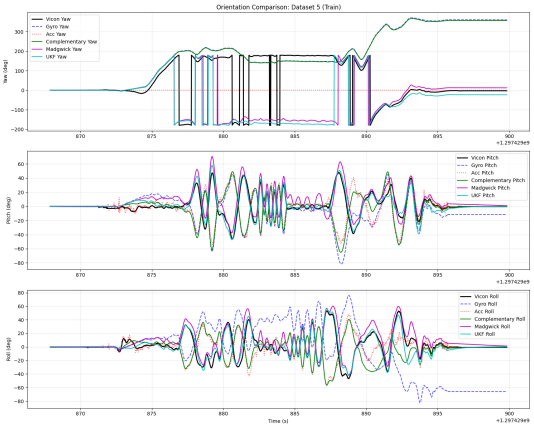

Fig. 6. Dataset 5

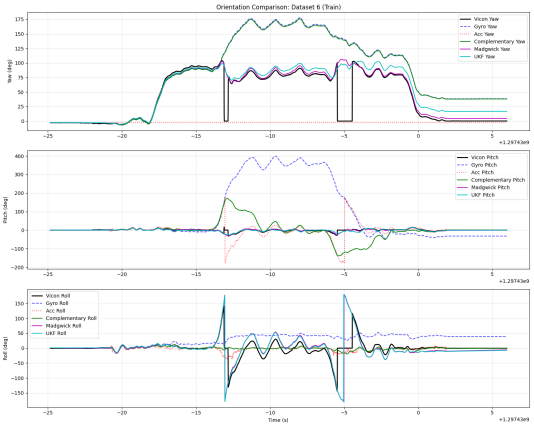

Fig. 7. Dataset 6

B. Test set - Orientation plots

The following Datasets were the Test sets that did not have the Vicon data accessible this was part of the imuRaw7,8,9,10 files although it is hard to verify their correctness, The videos from rotplot do show us that the UKF outperforms others and performs well!

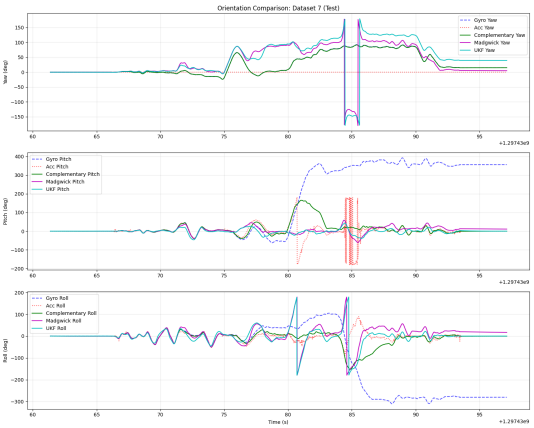

Fig. 8. Dataset 7

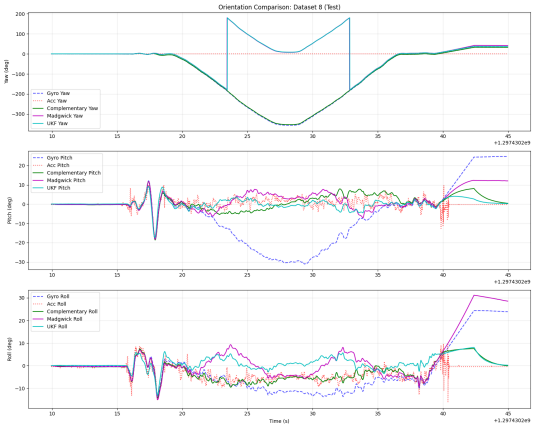

Fig. 9. Dataset 8

Fig. 10. Dataset 9

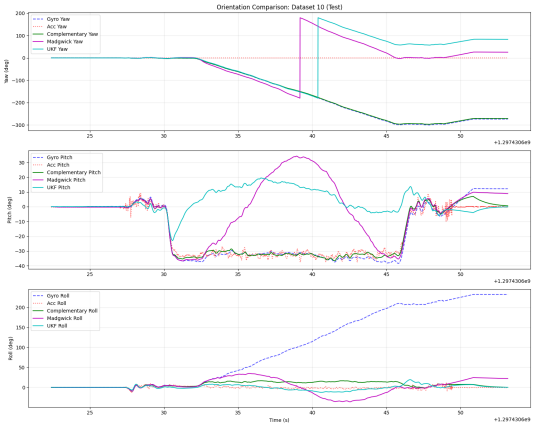

Fig. 11. Dataset 10

For the Training sets while some of the plot regions get tracked pretty well, there are definitely issues with the current tunes,We did fine tune for each of the dataset,where UKF outshines all the other methods, but in reality that is not how we implement thus we settle with a more general UKF value, thats Stable and works most of the time.

The test sets assumed that the initial orientations are Zeros, therefore that may play some role on some of the plots not following the trend, but considering the fact how accurate UKF is, We would still consider it working as well as it can. Performance does vary with tuning for sure. We could theoretically run a DL model on to find the perfect Q and R values, but that is out of the scope of this project.

C. Rotplot Videos

The Rotplot 3D visualizations videos were also made, for each of the Dataset. The video includes 6 sub-videos, showcasing the orientations from Gyro only, Acceleration only, Complementary Filter, Vicon, Madgwick Filter and UKF. These provide a really good idea of how good the implementation is running with visuals that are far more intuitive.

You can access the video links from the google drive here. [9]. (check the Citation).

REFERENCES

[1] Perception and U. o. M. Robotics Group, "Enae788m: Class 2 part 2 – imu basics, attitude estimation using cf and madgwick," YouTube, 2019, accessed: 2025-08-28. [Online]. Available: https://www.youtube.com/watch?v=8hRoASoBEwY

[2] Karooza, "Attitude estimation using imu sensors," Online Article, 2021, accessed: 2025-08-28. [Online]. Available: https://karooza.net/attitude-estimation-using-imu-sensors

[3] N. Sanket, "Attitude estimation using imu sensors (tutorial)," GitHub Pages, 2020, accessed: 2025-08-28. [Online]. Available: https://nitinjsanket.github.io/tutorials/attitudeest/imu.html

[4] MathWorks, "Understanding sensor fusion and tracking, part 1: What is sensor fusion?" YouTube, 2017, accessed: 2025-08-28. [Online]. Available: https://www.youtube.com/watch?v=6qV3YjFppuc

[5] ——, "Understanding sensor fusion and tracking, part 2: Fusing a mag, accel, gyro estimate," YouTube, 2017, accessed: 2025-08-28. [Online]. Available: https://www.youtube.com/watch?v=0rlvvYgmTvI

[6] B. Douglas, "Drone control and the complementary filter," YouTube, 2016, accessed: 2025-08-28. [Online]. Available: https://www.youtube.com/watch?v=whSw42XddsU

[7] M. Ulusoy, "Nonlinear state estimators — understanding kalman filters, part 5," Video, MathWorks, May 2017, published 18 May 2017. [Online]. Available: https://www.mathworks.com/videos/understanding-kalman-filters-part-5-nonlinear-state-estimators-1495052905460.html

[8] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," Conference Paper, IEEE International Conference on Rehabilitation Robotics (ICORR), 2011, [Online]. Available: https://drive.google.com/file/d/1LLJ5fZFMdWO4IAT66aDA-JIpTM7FhE47/view?usp=sharing.

[9] P. Katyal, A. Ramanathan, and H. Kortus, "Project files for rbe595 - project 1b - ukf," Google Drive, 2025, accessed: 2025-09-14. [Online]. Available: https://drive.google.com/drive/folders/12D0hPfsTfochTBFKuDVKrhR7JXtZdsr5?usp=sharing