RBE 595 - Advanced Navigation
Lab #3 - Nonlinear Filters
Jason Patel

Introduction

State estimation plays a critical role in mobile robotics and autonomous systems, especially for localization and navigation in uncertain environments. This report presents a comparative study of two widely used nonlinear state estimation methods: the Unscented Kalman Filter (UKF) and the Particle Filter (PF). The focus is on estimating the full 6-DOF pose (position and orientation) of a drone using visual and inertial sensor data.

In this lab we experimented with 2 types of filters to fuse inertial data from the IMU with measurement data from a PersepectiveNPoint model in order to perform localization of a Quadcopter.

Problem Description

The system aims to estimate the drone's pose using:

- AprilTag detections from camera images (providing sparse pose measurements)
- IMU data (accelerometer and gyroscope readings)

The dataset includes eight MATLAB files (studentdata0.mat to studentdata7.mat) each containing time-stamped measurements. Ground truth is provided via Vicon motion capture data.

State Space

A 15-dimensional state vector: Position (3), orientation (3), velocity (3), gyro bias (3), accel bias (3), was used to represent the state space.

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix}$$

Process Model

The process model is based on the acceleration output from the IMU. We account for the higher order dynamics of the drone (pitching and rolling) and account for the change in the IMU output with respect for the gravity vector.
The gyro and accelerometers are modeled with noise as follows:

$$\hat{\boldsymbol{\omega}} = \boldsymbol{\omega} + \mathbf{b}_g + \mathbf{n}_g$$

$$\hat{\boldsymbol{a}} = R(\mathbf{q})^T(\mathbf{a} - \mathbf{g}) + \mathbf{b}_a + \mathbf{n}_a$$

Where Uw= $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z] = \mathbf{u}_\omega$

And Ua= $\boldsymbol{a} = [\ddot{p}_x, \ddot{p}_y, \ddot{p}_z] = \mathbf{u}_a$

Continuous time process model:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ G(\mathbf{q})^{-1}\mathbf{u}_\omega \\ \mathbf{g} + R(\mathbf{q})\mathbf{u}_a \\ \mathbf{n}_{bg} \\ \mathbf{n}_{ba} \end{bmatrix}$$

, where Uw and Ua are the commanded angular velocity and linear acceleration, g is the gravity vector, and G(g) and R(q) are defined as follows:

$$G(\mathbf{q}) = \begin{bmatrix} \cos\theta & 0 & -\cos\phi\, sin\theta \\ 0 & 1 & \sin\phi \\ \sin\theta & 0 & \cos\phi\cos\theta \end{bmatrix}$$

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

Measurement Model

The measurement model is a simple vector plus measurement noise (which is omitted from the UKF filter).

$$\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \end{bmatrix} + \boldsymbol{\nu}$$

The h(x) function maps the measurement to the state space using a simple 6x15 matrix. A simple example as show here:

$$\mathbf{z} = \begin{bmatrix} \mathbf{I} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{I} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{q} \\ \dot{\mathbf{p}} \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} + \mathcal{N}(\mathbf{0}, \mathbf{R})$$

Methods

Both filters are implemented in Python. Key classes include:

- `MeasurementData`: Loads and processes AprilTag and IMU data and runs the measurement model
- `Localization`: Runs either UKF or PF over datasets and plots the results
- `UkfFilter` and `ParticleFilter`: Contain the specific implementations
- `Main:` Runs all experiments and outputs statistics and plot folder is filled with plots for orientation and position wrt ground truth; covariance calculations are turned off for the measurement model

Each filter estimates the state over time and compares against ground truth using:

- Trajectory plots
- Orientation plots (roll, pitch, yaw)
- RMSE calculations for position

Unscented Kalman Filter

Used the 15-dimensional state representation. Sigma points are generated using the Julier method from the original UKF paper. The measurement model is a linear mapping from the visual sensor model (April Tag locations in the global frame) using the H matrix. This method uses deterministic sampling of the sigma points to propagate uncertainty. The propagation model, or state transition model, does not include added noise directly, and is non-linear. Because the UKF algorithm does not require direct linearization of the state transition model, the f(x) of the propagation was left as a non-linear model in the f(x) function.
The covariance of the measurement model was originally taken from a calculated covariance matrix performed on matlab file 5, but was later replaced with a custom measurement covariance matrix which was calibrated to the best results.

Particle Filter

As the particle filter is a monte carlo based approach without a covariance matrix to introduce noise, the noise was added to the Ua and Uw vectors directly, and to the position and velocity

terms. The particle filter was evaluated with 250, 500, 750, 1000, 2000, 3000 particle filters in the experiment. The particle filter showed enormous potential as the 250 sigma point model outperformed the UKF filter.

RMSE

RMSE, root-mean-squared error, was used to compare measurement model's estimate (April tags) to the ground truth vicon data, and then further to compare the filters estimates to the vicon data, and then a simple comparison difference between the 2 to determine if the filtered results yielded additional accuracy, which I call RMSE Advantage.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i}^{n}(x_i - \bar{x_i})}\,)$$

Results

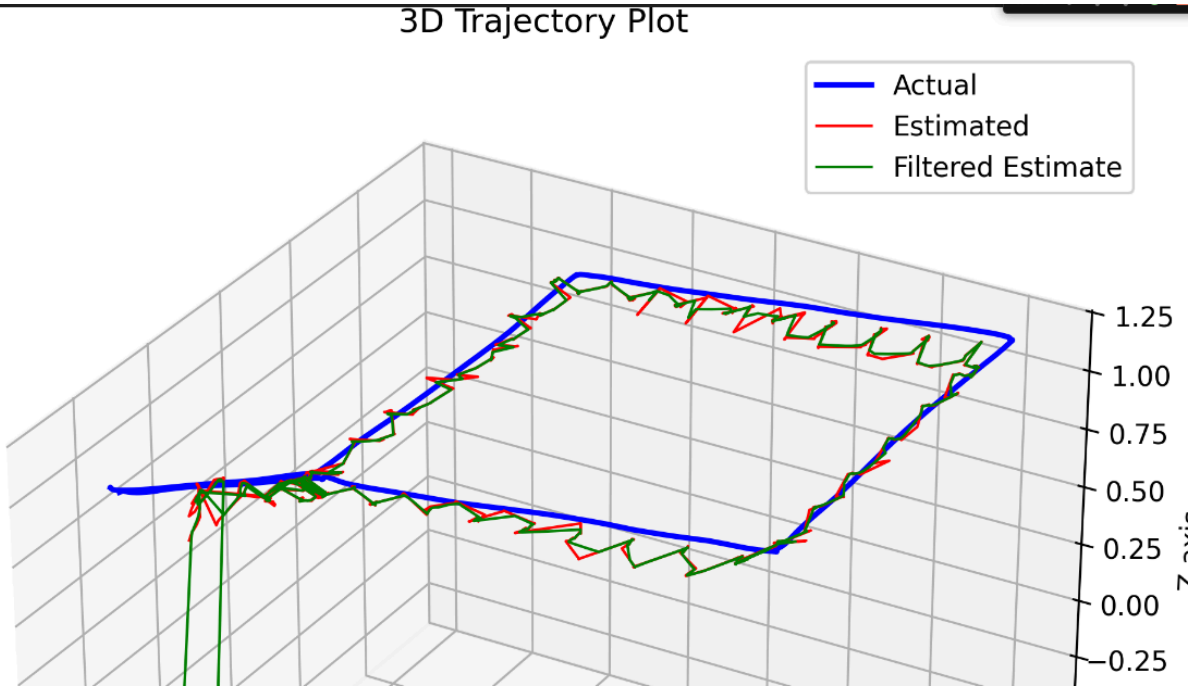
Trajectory and orientation plots are saved in `./plots/`

UKF Results

- Run for 7 out of 8 datasets because the first matlab file did not contain IMU data
- Produces stable, however somewhat jagged, trajectory estimates
- See below stats for discussion:

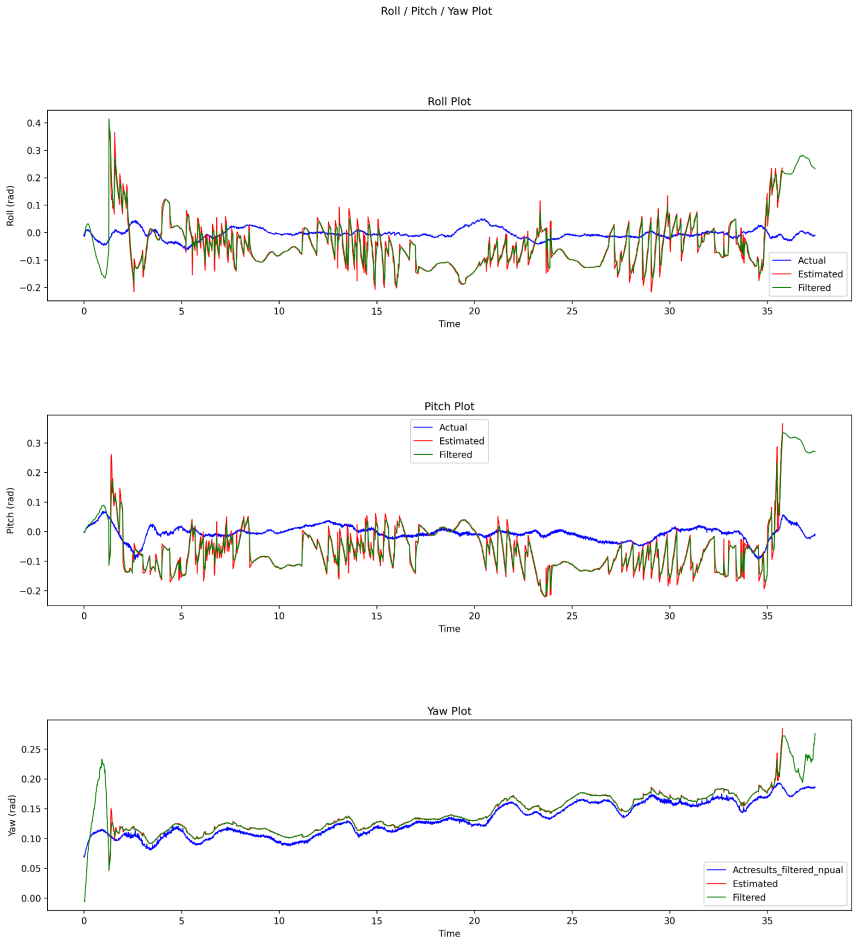
```
(venv) (base) mind@NU25:~/dev/rbe/AdvNav_UKF_ParticleFilters$  /usr/bin/env /home/mind/dev/rbe/AdvNav_UKF_ParticleF
tensions/ms-python.debugpy-2025.8.0-linux-x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 49433 -- src/main
RMSE of rmse_filtered:  0.35570485135363594
RMSE of rmse_measurement_model:  0.3590358950262903
RMSE of rmse_filtered:  0.355713612275884
RMSE of rmse_measurement_model:  0.3685339184500923
RMSE of rmse_filtered:  0.3343265207018101
RMSE of rmse_measurement_model:  0.339885540967506
RMSE of rmse_filtered:  0.34951717475881594
RMSE of rmse_measurement_model:  0.3606399324991077
RMSE of rmse_filtered:  0.4306166366551924
RMSE of rmse_measurement_model:  0.6522614033251368
RMSE of rmse_filtered:  0.4092771121061863
RMSE of rmse_measurement_model:  0.5871721447558709
RMSE of rmse_filtered:  0.401709558038398
RMSE of rmse_measurement_model:  0.6143745598644883
RMSE results for all datasets with UKF Filter: 0.3766950665557033
```

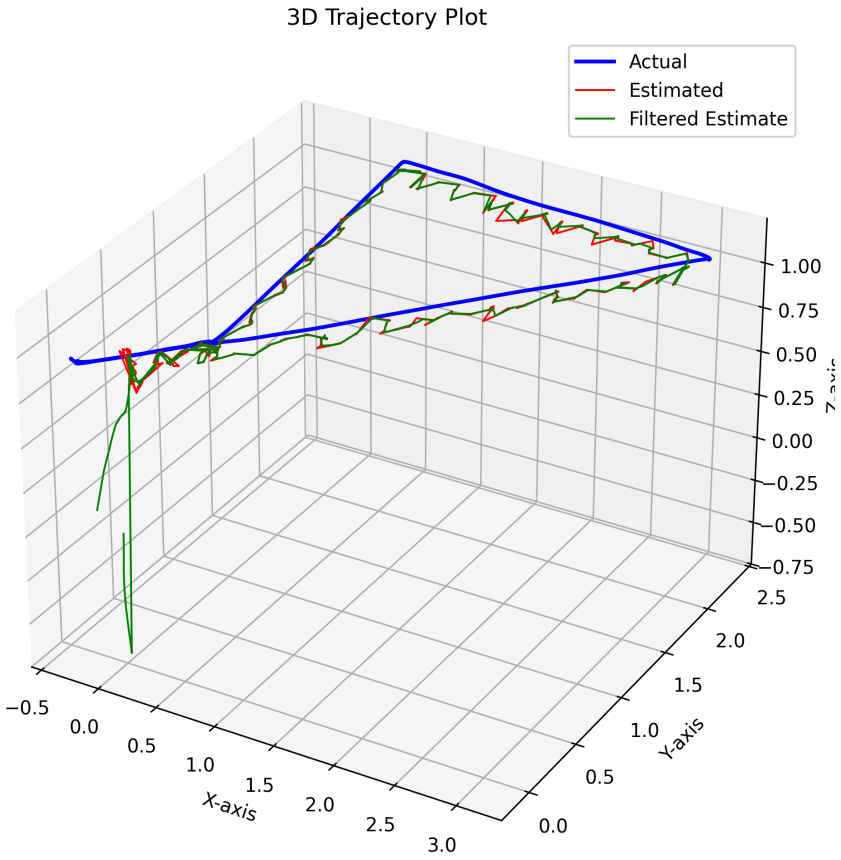
Here we can see that the average RMSE across the 7 valid datasets is 0.3767m. In all matlab files the filtered RMSE outperformed the measurement models' RMSE, which shows that the UKF filter was able to incorporate the motion model using the IMU data to improve the estimates.



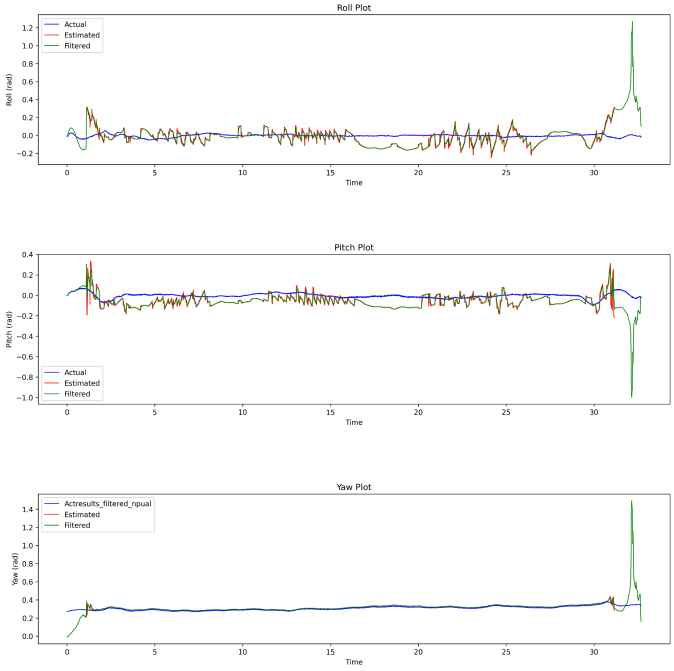
This figure shows the UKF closely tracking ground truth for file 5. On the left side you can see it not rising as high for the measurement jumps, rather it smoothes out the trajectory and is closer than measurement itself. The start or end of the session shows jumps which reduced the accuracy in the RMSE calculation.

On the orientation plot to the right (file 5), we can see the “Filtered” plot closely tracking the measurement model’s orientation.

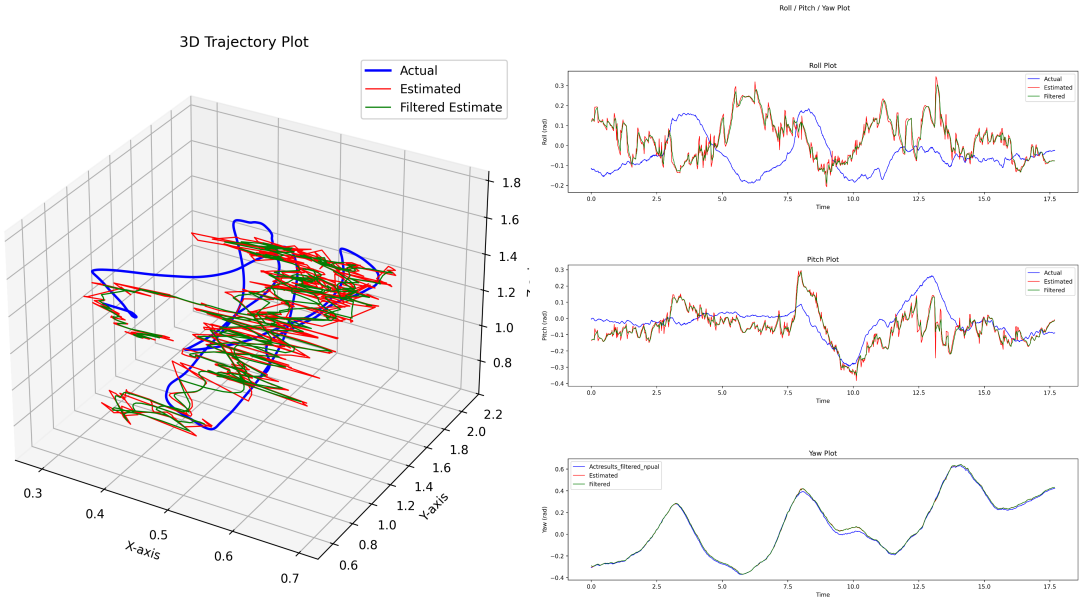




On the plot from the UKF, for file 7, we can see the Filter tracking the actual closely and smoothing out the measurement model's estimates.



Here on the plot to the left for file 7, we see the orientation tracking closely, except for the jumps at the beginning and the end of the data series.

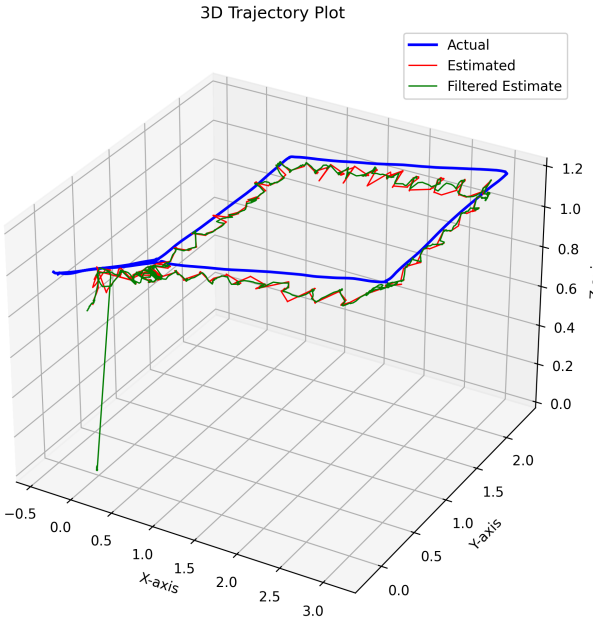


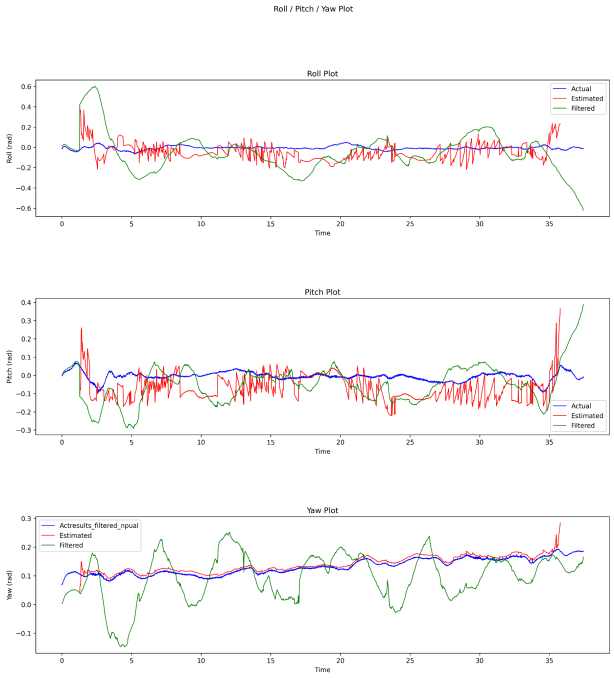
As shown above, for file 1, the UKF filter is tracking the vicon data, however, not as closely as some of the other files. The orientation plot on the right closely tracks the yaw. The pitch tracks but seems to have some sort of bias. The roll seems shifted.

Particle Filter Results

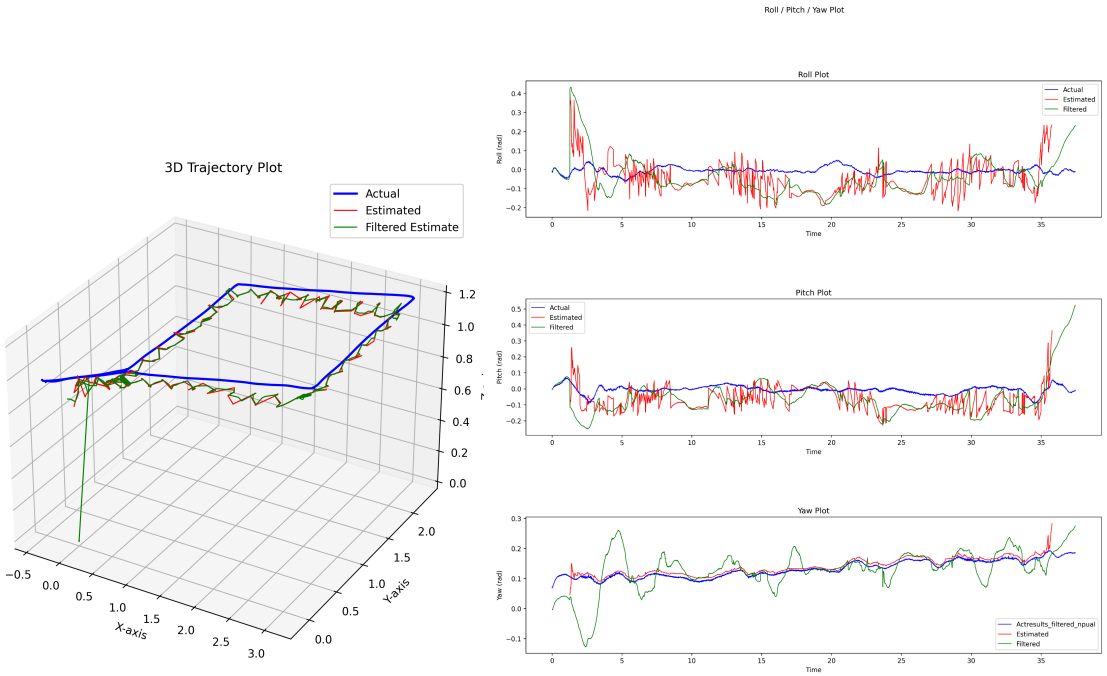
- Run for 7 out of 8 datasets because the first matlab file did not contain IMU data
- Produces trajectory estimates which follow the measurement model with some improvement

```
RMSE of rmse_filtered:  0.3589221191944724
RMSE of rmse_measurement_model:  0.3590358950262903
RMSE of rmse_filtered:  0.354782596169638
RMSE of rmse_measurement_model:  0.3685339184500923
RMSE of rmse_filtered:  0.33460678991171283
RMSE of rmse_measurement_model:  0.339885540967506
RMSE of rmse_filtered:  0.3479801034574052
RMSE of rmse_measurement_model:  0.3606399324991077
RMSE of rmse_filtered:  0.4138556139707379
RMSE of rmse_measurement_model:  0.6522614033251368
RMSE of rmse_filtered:  0.396934051038942
RMSE of rmse_measurement_model:  0.5871721447558709
RMSE of rmse_filtered:  0.38832149895382606
RMSE of rmse_measurement_model:  0.6143745598644883
RMSE results for all datasets with Particle Filter [250]: 0.3707718246709621
```





Above, the plots from figure 5 of the Particle Filter-250 run.



Above, the plots from figure 5 of the Particle Filter-4000 run. The orientation plot seems to smooth out the measurement model's estimates (except for Yaw)

Particle Filter RMSE Results for File 7

Particle Filter Count	RMSE-Particle Filter	RMSE-Measurement Model
250	0.38832149895382606	0.6143745598644883
500	0.38859125111412884	0.6143745598644883
750	0.3890422555533071	0.6143745598644883
1000	0.3880234514916712	0.6143745598644883
2000	0.38769390928105957	0.6143745598644883
3000	0.38773689825597885	0.6143745598644883
4000	0.388472719371678	0.6143745598644883
5000	0.38808697044469453	0.6143745598644883

Summary of Results

Metric	UKF	PF
Model Assumption	Gaussian	Nonparametric
Computation	Efficient	Scales with particles
Accuracy	Good for mild nonlinearity	Better for high nonlinearity
RMSE Stability	High	Varies with particle count

UKF is efficient and suitable when the noise is approximately Gaussian and system nonlinearity is moderate. PF offers better flexibility and accuracy under high noise and nonlinearity but at a computational cost.

Comparison

The UKF filter executes quickly, is easier to tune, but much harder to implement. In fact, the first implementation was originally done using a library, filterpy, before it was rewritten to implement the algorithm from scratch. However, the results were not as good as the Particle filter implementation. Generating the sigma points was an extra step, but resembled the Particle filter's implementation in that the sigma points, like the particles, are generated and then the motion model propagates based on the kinematics/dynamics. Just the method for generating the sigma points and the particles are different. The UKF Filter's generation of sigma points had issues calculating the square root of the covariance matrix and special care had to be done to handle this challenge. The UKF filter adds noise in the prediction step to the P matrix, whereas the Particle filter adds noise directly into the motion model/propagation step to the velocity, position, and acceleration directly.

The Particle filter was a challenge to get it to converge and until the motion and measurement models were implemented correctly and the noise terms added the particle filter didn't look like it would converge. However, after correcting the implementation and adjusting the noise parameters, the 250 particle filter outperformed the UKF in terms of accuracy via the RMSE. The only major challenge would be to address the performance of the Particle filter as it increases with the particle size. Increasing the particle count improved performance up to a certain point, as the 4000 particle run actually did not outperform the 2000 particle experiment. Additionally, more work could be done to improve the Particle filters tracking of the orientation. Finally, a great result for both filters can be seen on File 7 in that the measurement model produced a RMSE of .614m and both filters were able to improve the accuracy in their respective estimates by almost .3m! My only disappointment is that both filters seem unable to produce RMSE accuracies below .3m given the dataset and sensor measurements provided.

Conclusion

Both the UKF and PF can accurately estimate 6-DOF pose using camera and IMU data. In this experiment the Particle Filter provided the best results. The UKF provides a reliable and efficient approach for real-time applications, while the PF delivers higher accuracy with a much higher computational resource cost. The choice depends on the application constraints and noise characteristics.