Project 2: Orientation Estimation
ESE 650 Learning in Robotics, 2018 Spring

Dian Chen

March 1, 2018

1 Introduction

Kalman Filtering is widely used in state estimation. However, it cannot be applied to systems with non-linear dynamics or measurement model. The Unscented Kalman Filter is one type of approximation that tries to capture and maintain certain statistics of the non-linear system, basically by introducing sigma points.

In this project, we are trying to estimate poses, which have non-linear dynamics (Lie algebra) during transition. I chose to use a 4-dimensional state vector which contains the 4 components of the quaternion representing the pose, and use gyro data (angular velocity) as inputs, accelerometer data as measurement. An Unscented Kalman Filter was applied to estimate the sequence of poses and the results were used to generate panorama stitches given corresponding images.

(Notice: I made some modifications to the plot functions of the original code to deal with datasets without Vicon data, but I didn't submit the final code since the estimation part is untouched. I'm just noticing you that the plots generated using the submitted code will be slightly different from these final plots.)

2 Model Selection

2.1 Motion Model

In this project, I chose to use the quaternion pose representation as the state vector, and the gyro data, i.e. angular velocity measurements as inputs. The non-linear motion model of the system can be described as follow:

$$q_{k+1} = g(q_k, \omega_k, \xi_k)$$

More specifically, in each tiny time interval we can view the angular velocity as constant which causes the body to rotate a little:

$$q_{k+1} = q_k q_\xi q_\Delta$$

The gyro measurements are unavoidably noisy that some may doubt if they can be used directly as inputs. In my perspective of view, the uncertainty in our "inputs" has been considered in the noise term η_k, which packs all uncertainties during the motion (input noise, motion noise, etc.). So using the gyro data as inputs is a reasonable choice.

2.2 Measurement Model

The accelerometer measurements come from another non-linear process which is projecting the gravity vector in the inertial frame (world) into body frame:

$$z_k = R_k^T[0, 0, \mathbf{g}]^T + \eta_k$$

3 State Estimation

3.1 Data Preprocessing

The raw data have biases and haven't been properly scaled. So the first step of our state estimation is to convert the raw data to physically meaningful values. For biases, I inspected the first few hundreds frames of raw data, and decided to take the mean of first 200 frames (which basically correspond to stationary, upright pose) as biases for both gyro and accelerometer. After subtracting the biases, the data were scaled using equations and metrics given in the manual. Finally, I calculated and added the correction term to the third component of accelerometer measurements to make them have a valid "gravity".

To get a valid number of time intervals, the first frame of IMU data was not used. Moreover, for some dataset in which some evident translation was introduced (possibly putting the camera down), the IMU measurements don't match pure rotations. So I selectively discarded the last few hundreds frames to make the results look reasonable.

As for the time synchronization, I chose to match up the closest timestamps, which yields pretty good results.

3.2 Gyro Only

If we use only gyro data, meaning that we use only motion update, then we simply perform the successive small rotations and integrate over all the small time intervals:

$$q_{k+1} = q_k q_\Delta$$

in which

$$q_\Delta = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\frac{\omega}{|\omega|}], \theta = |\omega\Delta t|$$

Since I treat the gyro data as known inputs, the noise is taken care in the noise covariance matrix, which is actually not used in the motion-only update equation.

3.3 Accelerometer Only

In this track, I first applied a low pass filter to the original accelerometer measurements:

$$z_k = \alpha z_k + (1 - \alpha) z_{k-1}$$

After this initial, simple filtering, the pose is calculated as follow:

$$\theta = \arctan(-Acc_x, Acc_y)$$

$$\phi = \arctan(Acc_y, \sqrt{Acc_x^2 + Acc_z^2})$$

Here θ and ϕ correspond to pitch and roll angles, respectively. Notice that with only accelerometer data, we are unable to retrieve the yaw angle but only pitch and roll. But we can still compare these two angles with the ground truth.

3.4 UKF Steps

As suggested in the Kraft paper, I implemented the UKF in quaternion space and mostly manipulated sigma points:

1. Initialize the state vector $\hat{x}_0$ and covariance matrix P_0, as well as motion noise matrix Q and measurement noise matrix R.
2. Add Q and P_{k-1} and transform this noise matrix to 6 3-D $\{W_i\}$ vectors.
3. Shift $\{W_i\}$ by $\hat{x}_{k-1}$ to generate the sigma points $\{X_i\}$.
4. Apply the motion model on $\{X_i\}$ to get the sigma points $\{Y_i\}$ after motion transform.
5. Compute the quaternion mean (I chose to use gradient descent) of $\{Y_i\}$ to get $\hat{x}_k^-$.
6. Remove the quaternion mean from $\{Y_i\}$ to get $\{W_i'\}$.
7. Compute the a-priori process covariance P_k^- using $\{W_i'\}$.
8. Apply the measurement model on t$\{Y_i\}$ to get the expected measurement $\{Z_i\}$ for each transformed sigma points.
9. From $\{Z_i\}$ compute the mean z_k^- and then covariance P_{zz}.
10. Compute the innovation ν_k from z_k^- and actual accelerometer data z_k.
11. Compute innovation covariance $P_{\nu\nu}$ and cross correlation matrix P_{xz}.
12. Compute the Kalman gain K_k using all these covariance matrices.
13. Finally apply the weighted innovation $K_k\nu_k$ to $\hat{x}_k^-$ and yield the estimation $\hat{x}_k$, and compute the estimated error covariance P_k. This brings this iteration to an end and go back to 2. again. All these steps were implemented in the codes with detailed commenting.

4 Panorama Stitching

In this project I chose to utilize spherical coordinates to do the panorama stitching, which is a simple approach. To cover the π and 2π angles in spherical coordinate system, I set the size of the stitched image to be $(H, 2H)$, and used $H = 540$ in the implementation. Each image from the real camera is viewed as from a field of view (FOV) of $(\pi/4, \pi/3)$. The stitching pipeline goes as follow:

1. Compute the polar angle and azimuthal angle of each pixel, and compute their X, Y, Z coordinates by setting a constant radius (1 in the implementation), meaning that the pixels are now evenly distributed on the sphere.

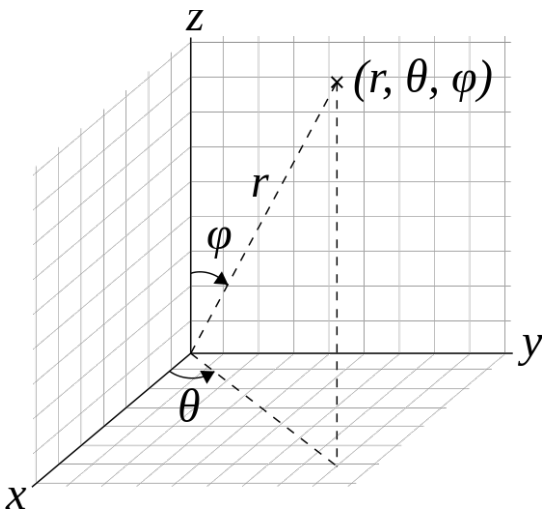

Figure 1: Spherical coordinate system. From Wikipedia

2. Apply the estimated pose R to each pixel, to compute the transformed position.
3. Convert the transformed X, Y, Z coordinates back to spherical coordinates θ and ϕ.
4. Scale θ and ϕ to H and $2H$, which correspond to the coordinates on the stitched image (canvas).
5. Add some constant value to each coordinates to make the identity pose centered properly on the big canvas, and finally copy the pixels onto the big canvas.

5 Results

All the videos and plots are available at this google drive link:

`https://drive.google.com/drive/u/0/folders/1ZaK1VYg9BuCkj78rUt05ci32OjCe0mOj`

5.1 State Estimation

Since we're estimating pose, we should first convert the pose to Euler angles to compare the results. The conversion from quaternions to rotation matrices and from rotation matrices to Euler angles are implemented in the code. There are four possible pose sources: from gyro only, from accelerometer only, from UKF estimation and from Vicon (viewed as ground truth). The curves of all training and test data are as follow (dataset 1 to 10 have Vicon data, dataset 11 to 13 don't):

5.2 Panorama Stitching

For dataset 1, 2, 8 to 13, they come with camera images. The generated videos can be found in the google drive. Here's two example screen shots of the panorama (from 8 and 10).

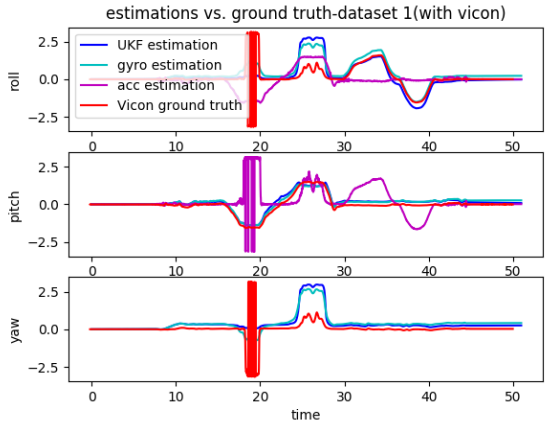

Figure 2: Euler angles, dataset 1(with vicon)

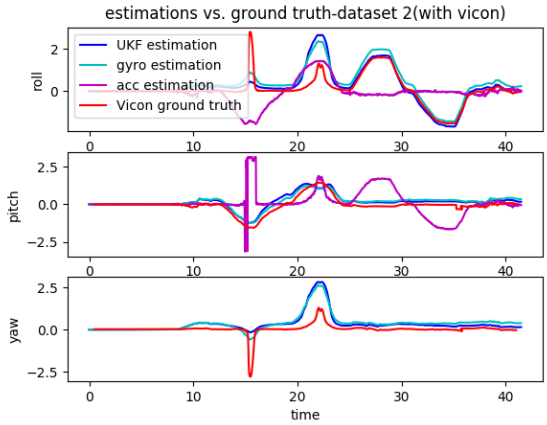

Figure 3: Euler angles, dataset 2(with vicon)

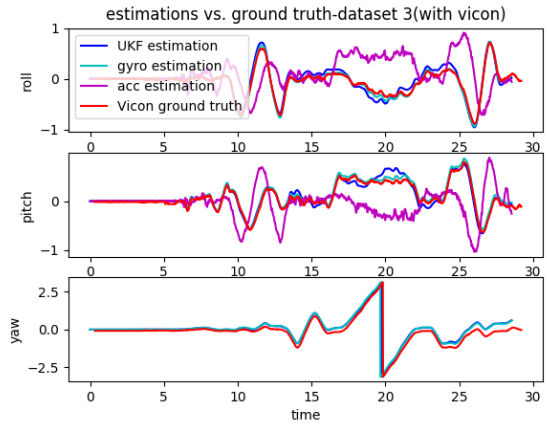

Figure 4: Euler angles, dataset 3(with vicon)

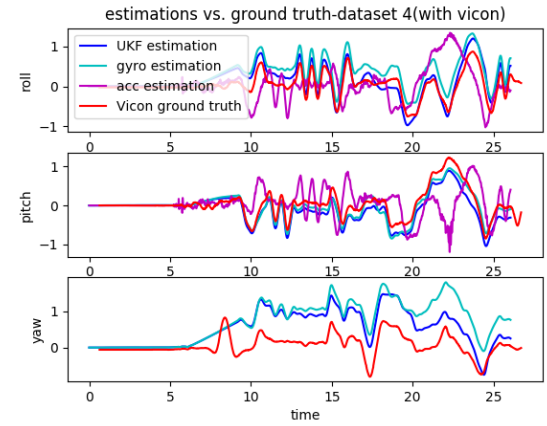

Figure 5: Euler angles, dataset 4(with vicon)

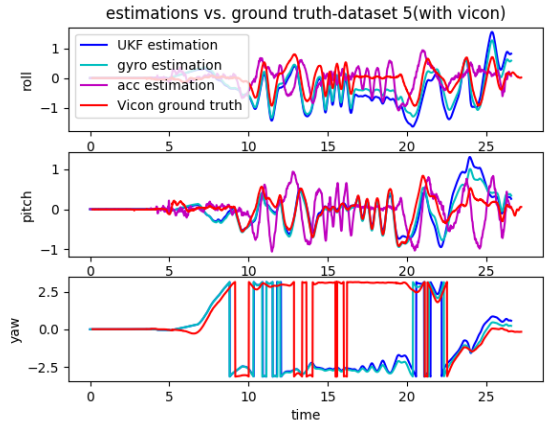

Figure 6: Euler angles, dataset 5(with vicon)

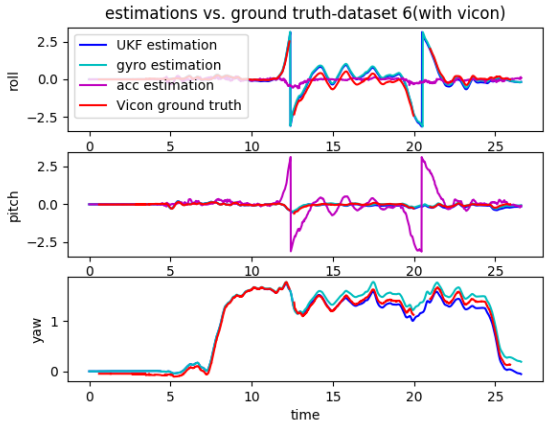

Figure 7: Euler angles, dataset 6(with vicon)

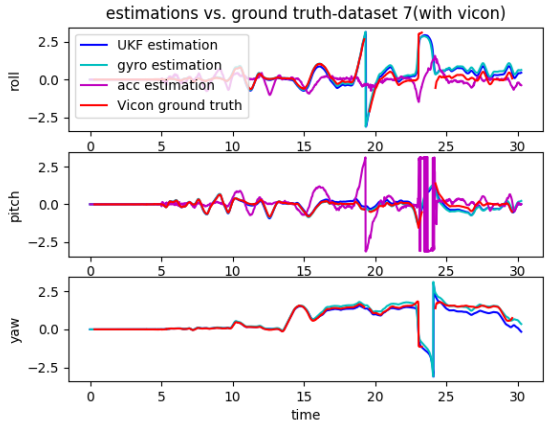

Figure 8: Euler angles, dataset 7(with vicon)

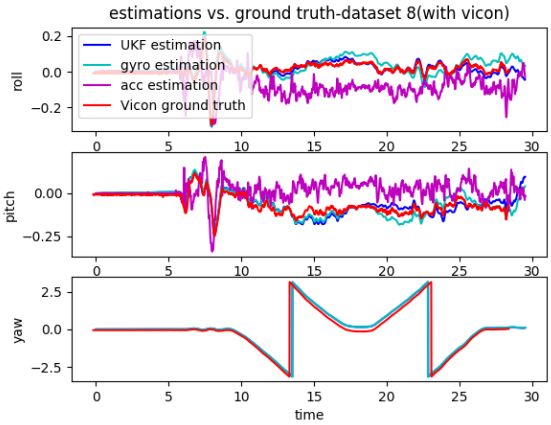

Figure 9: Euler angles, dataset 8(with vicon)

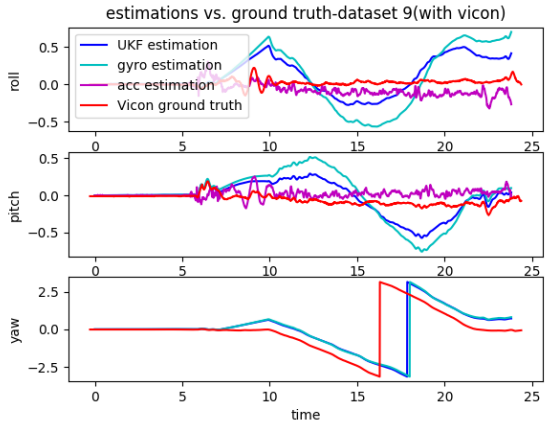

Figure 10: Euler angles, dataset 9(with vicon)

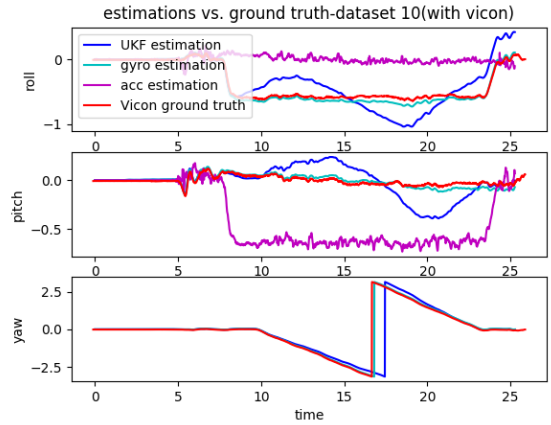

Figure 11: Euler angles, dataset 10(with vicon)

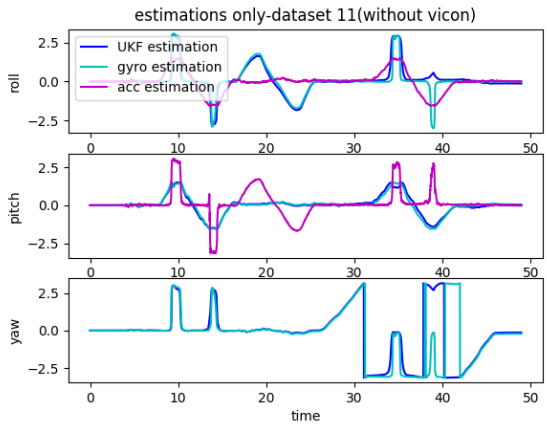

Figure 12: Euler angles, dataset 11(without vicon)

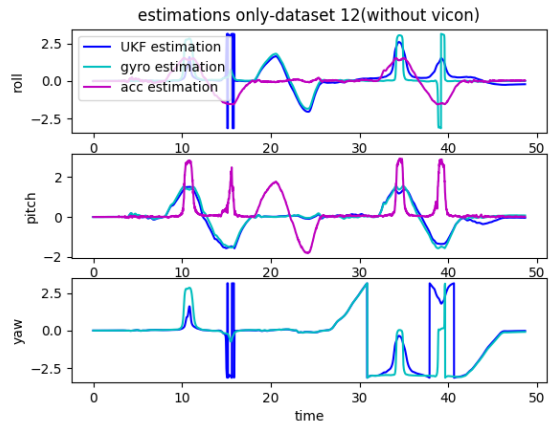

Figure 13: Euler angles, dataset 12(without vicon)

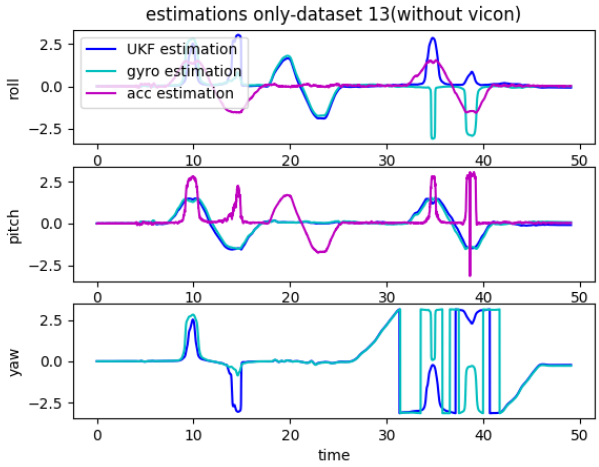

Figure 14: Euler angles, dataset 13(without vicon)

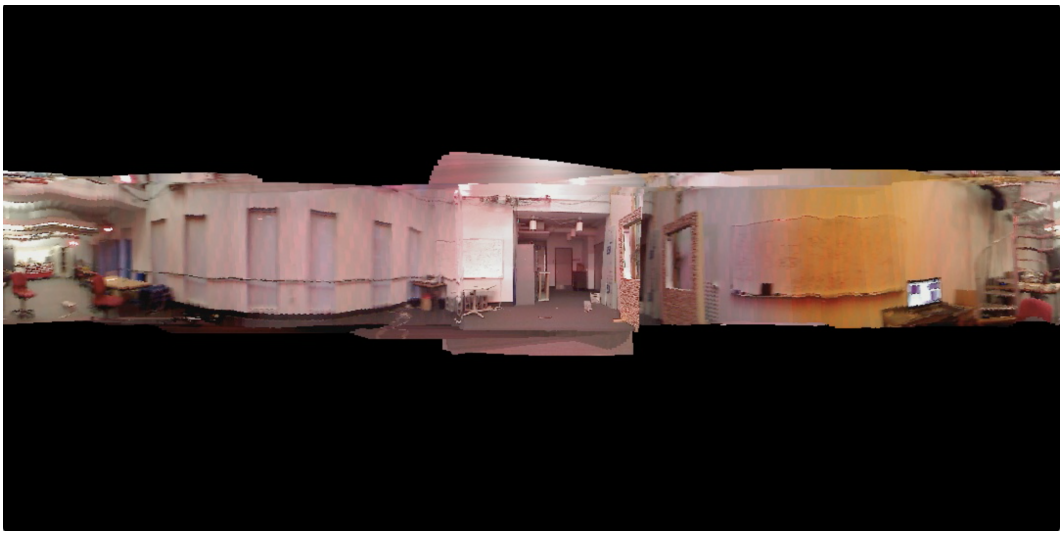

Figure 15: panorama, dataset 8

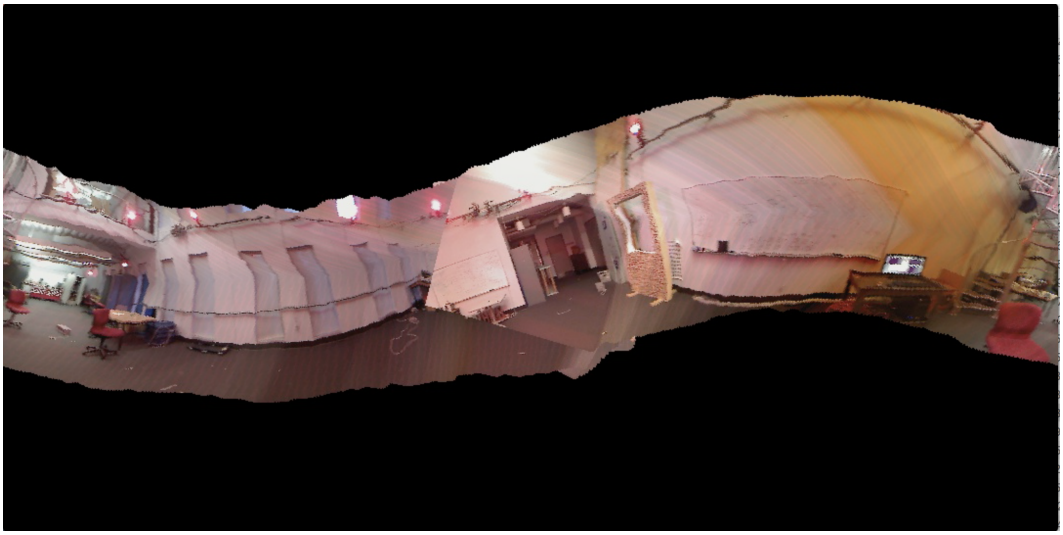

Figure 16: panorama, dataset 10

6 Discussion

6.1 Data Preprocessing

In this project I simply used the average of the first 200 data frames as the mean, which is a constant value after this preprocessing is done. In practice we may need to estimate online the biases to suppress angular velocity drift as much as possible. As for timestamps synchronization, the approach I used in this project is also relatively naive and in practice some more sophisticated way may be applied, such as interpolation.

6.2 Comparison Between UKF, Gyro and Accelerometer

By only integrating the gyro measurements or computing Euler angles directly from the accelerometer data, we can actually get surprisingly good results, as we can see in some plots their curves agree with the Vicon ground truth. And the results from only gyro measurements are generally better than those from only accelerometer. I think this is because the gyroscopes have better response to high frequency whereas the accelerometers response better to low frequency (due to the underlying mechanics), so when the ground truth curve suddenly bumps up or heads down (sudden movements), the gyroscope readings can follow up more easily. However, both these two alone cannot be trusted entirely, since their readings are very noisy and simple integration without correction will eventually drift away.

As for the UKF estimation, we can see from the plots that it produced better results than both naive direct estimation approach, and the panorama stitching are generally good (especially those with curves closely following the ground truth). By using UKF, we're effectively utilize both the motion model and actual measurements, as well as taking into account the non-linear nature of the system. So the UKF is obviously less prone to noise that causes most of the drift.

However, we can also see that some parts of the UKF curve do not follow the ground truth satisfac-

torily, and during panorama stitching I found two major factor that greatly affects the performance: 1) The way of computing the mean of quaternion. I tried both approach and the iterative gradient descent gave better performance than principal component. I'm still trying to figure out why and the difference between these two approaches. 2) The initialization of error covariance P_0, and choice of noises Q and R. The stitching results look much more reasonable if I set them to larger values, which might mean that I'm being more modest with my estimation and measurement accuracy.

6.3 Other Comments

There are still other facts that affects the performance. When the body is not doing pure rotation (as in the last frames in some datasets), some parts of the panorama stitching would not match up even though the estimation of rotation is good enough. Also, if the body has evident acceleration then the measurement model would not be the simple mapping of gravity vector from world to body frame. These two factors come from the data side and there's nothing UKF can do to avoid them. However, there are still other things to do to improve the UKF, which I'm interested in.

One last comment on the crazily flipping angles occurred in the plots. This is because the actual angle is near π and causes the numerical results to flip between π and $-\pi$. In practical implementations we should be careful of this kind of wild fluctuation.

Reference

1. Kraft, Edgar. "A Quaternion-based Unscented Kalman Filter for Orientation Tracking."
2. `https://en.wikipedia.org/wiki/Spherical_coordinate_system`
3. `https://theccontinuum.com/2012/09/24/arduino-imu-pitch-roll-from-accelerometer/`