Using Quaternion-based UKF for Orientation Tracking and Panorama: A Project Review

Yue Meng
Department of Electrical Computer Engineering
University of California, San Diego
yum107@eng.ucsd.edu

***Abstract*—This paper presented approaches using Quaternion-based Unscented Kalman Filter for orientation tracking problems. In this work, we basically followed the idea in [1] to calibrate the orientation. Our goal is to use inertial measurement unit readings to tracking the camera orientation and generate panorama. The result showed that this was a nice try.**

***Index Terms*—Orientation Tracking, Unscented Kalman Filter, Quaternion, Panaroma**

I. Introduction

Orientation tracking, which is to calibrate the pose of a particular object using sensor data or images, is a very important domain nowadays. It is the fundamental compomnent in SLAM problem. Various of applications can be found in medical research, mining robots and autonomous driving.

The essential part of orientation tracking problem is how to implement the calibration. As for rigid rotation object, the challenge comes from the fact that the motion model and observation model are basically related with rotation process, and it is theoretically non-linear. Traditional kalman filter won't work because all its assumption is based on linear model and gaussian distribution.

Normal solution for this situation is try to force noise distribution to Gaussian and find simple way to calculate the mean and covariance for the state vector and the state-measurement vector. Using taylor expansion for approximation of motion and measurement model comes from Extended Kalman Filter, which trying to linearize the model from mean points. Sampling sigma points to calculate new possible mean and covariance results in Unscented Kalman Filter, which equivalent to adopting numerical ways to represent the origin distribution of noise. It hard to say generally which one is superior to another approach. In [2] the author claimed that when model is very likely to be piece-wise linear, EKF should work well. If the model contains more significant higher order moment, then UKF deserves a better result.

In our case, when facing a rigid rotation object, it is not a good idea to use Jacobian matrix to approximate the model, so we are going to use UKF for calibration. Since the state is just rotation, we would like to use represent the state in quaternion as its concise for computation and property of one-one-mapping. The rest of paper is arranged as follows. First we give the detailed formulations of oriented tracking problem in Section II. Techinical approaches are introduced in Section III. And at last we setup the experiment, results are presented in Section IV.

II. Problem Formulations

A. Tracking Problem

The object we are going to tracking is a rotating camera where the location is fixed and the orientation is varied based on its angular velocity. The motion model $A(x_k, u_k, w_k)$ can be written as

$$x_{k+1} = x_k \oplus \exp(0, \frac{u_k \cdot \Delta t_k}{2}) \oplus \exp(0, \frac{w_k}{2}) \tag{1}$$

where x_{k+1} and x_k is the state in time $k+1$ and k respectively, in quaternion form. u_k is the 3-dimensional angular velocity and w_k is the noise. $\oplus$ called "smart plus" is the way to represent quaternion multiplication, eg: $q \oplus p = q \cdot p = [q_s p_s - q_v^T p_v^T, q_s p_v + p_s q_v + q_v \times p_v]$. The exponential function $exp()$ is for quaternions operation. And δt_k is the time gap between time $k+1$ and k.

From the sensor equipped on the camera we can get the angular velocity $\{u_k = (u_{x,k}, u_{y,k}, u_{z,k}) \in \mathcal{R}^3\}$ and acceleration $\{g_k = (g_{x,k}, g_{y,k}, g_{z,k}) \in \mathcal{R}^3\}$. We adopt the right-hand coordinate where x-axis is the front, y-axis is the left side and z-axis is the top of the camera. The measurement model $H(x_k, v_k)$ indicates the relationship between the camera orientation and the acceleration

$$z_{k+1} = (x_k \oplus v_k) \oplus g \oplus (x_k \oplus v_k)^{-1} \tag{2}$$

where g is the gravity in quaternion form $q = [0, 0, 0, -1]$ and v_k is the measurement noise.

A dataset consist of timestamp and rawdata is denoted as $\mathcal{D}_{raw} = \{(t_i, D_{raw,i})\}$, where $D_{raw,i} = (d_{Ax,i}, d_{Ay,i}, d_{Az,i}, d_{Wx,i}, d_{Wy,i}, d_{Wz,i}) \in \mathcal{R}^6$ is the rawdata from sensor. And a preprocess model $\mathcal{F}$ given by datasheet is to transform rawdata $D_{raw,i}$ to sensible data in our problem $D_i = (g_{x,i}, g_{y,i}, g_{z,i}, u_{x,i}, u_{y,i}, u_{z,i}) \in \mathcal{R}^6$.

Then the problem can be presented as follows, considering state quaternion x_k, observation vectorz_k, given motion model A, observation model$H$, preprocessing model$\mathcal{F}$ and an initial guess as x_0 and assumption of both noises coming from independant gaussian distribution, using data ranging in timestep 0 to k from dataset $\mathcal{D}$ to calibrate the state vector $\{\hat{x}_{k+1}\}$. And at last we will derive a calibrated state vector series $\{\hat{x}_i\}$

B. Panorama Problem

The panorama problem is that: given a set of images $\{I_k \in \mathcal{R}^{h\times w\times 3}\}$ (h and w is the number of rows and columns for the image) and their corresponding orientation data $\{u_k = (u_{k,x}, u_{k,y}, u_{k,z}) \in \mathcal{R}^3\}$ and depth assumptions $\{Depth_k \in \mathcal{R}^{h\times w}\}$, try to reconstruct the 3D RGB data $\mathcal{P} = \{(x, y, z, (R, G, B))\} \in \mathcal{R}^{X\times Y\times Z\times 3}$ in real world frame, and at last project to a camera plane to generate a new image $\mathcal{I} \in \mathcal{R}^{H\times W\times 3}$(where H and W are new counts of rows and columns for the reconstructed panorama).

III. Technical Approaches

In this section we discuss about algorithms and models used in this project.

A. UKF for Orientation Tracking

Unscented Kalman Filter uses two steps performing iterally throughout all the sensor data by time order to get the calibration result. In the prediction step, we generate sigma points and integrate motion model to get mean and covariance for the state quaternion. In the update step, we use that predicted state quaternion to estimate the mean and covariance for the measurement vector (also the covariance by measurement vector and state quaternion). And use the difference (innovation) from this estimate measurement vector to the real sensor data to calculate the kalman gain, which lead us to assign proper weights for estimation and measurement to get more accurate mean and covariance for the state quaternion.

1) Prediction Phase: The inputs in prediction phase at time step t are previous calibrated state quaternion $x_{t|t}$, previous estimated covariance matrix $P_{t|t}$, control input u_t, constant covariance matrix of noise Q. Although the state is quaternion, with the constraints of norm as one, it only has three degrees of freedom. Hence the $P_{t|t}$ is 3-by-3 matrix to illustrate covariance in rotation-vector sense. Details can be found in afterwards calculation.

First of all we need to generate the sigma points $\{\mathcal{W}_i\}$ and their weights for mean $\{\alpha_{m,i}\}$ and for covariance $\{\alpha_{c,i}\}$. The first sigma point $\mathcal{W}_0 = 0$

$$\mathcal{W}_i = \pm\big(\sqrt{2n\cdot(P_{t|t}+Q)}\big)_{|(i-1)/2|}\,,\ i = 1, 2...2n \tag{3}$$

$$\alpha_{m,0} = 0\ \ \alpha_{m,i} = 1/2n\,,\ i = 1, 2...2n \tag{4}$$

$$\alpha_{c,0} = 2\ \ \alpha_{c,i} = 1/2n\,,\ i = 1, 2...2n \tag{5}$$

where $()_i$ is the i-th col for the square root matrix, and $|.|$ is the rounding operator. We add covariance from state and from motion model together to integrate noise to the sigma points. In the implementaion we use Cholesky decomposition for this step. n is the degree of freedom of state hence is 3.

Then we transform the sigma points to quaternion form.

$$q_{\mathcal{W}_i} = \exp\left[0, \mathcal{W}_i/2\right] \tag{6}$$

Next we can calculate the state sample points using motion mode$A(x_{t|t}\oplus q_{\mathcal{W}_i}, u_t, 0)$

$$\mathcal{X}_i = x_{t|t}\oplus q_{\mathcal{W}_i}\oplus\exp([0, \frac{1}{2}u_t\Delta t])\,,\ i = 0, 1, 2...2n \tag{7}$$

Because we have already add noise in the Cholesky part, so here we don't put noise in the motion model. Then we calculate the mean and covariance for state quaternion.

$$x_{t+1|t} = \sum_{i=0}^{2n}\alpha_{m,i}\mathcal{X}_i \tag{8}$$

And we need to pick the rotation part from the mean quaternion to calculate the covariance.

$$[0, e_{v,i}] = 2\log\left(x_{t+1|t}^{-1}\oplus\mathcal{X}_i\right),\ i = 0, 1, 2...2n \tag{9}$$

$$P_{t+1|t} = \sum_{i=0}^{2n}\alpha_{c,i}e_{v,i}e_{v,i}^T \tag{10}$$

By this point, we have got the estimated state quaternion $x_{t+1|t}$ and the covariance matrix $P_{t+1|t}$

2) Update Phase: The inputs in update phase is the estimated state quaternion $x_{t+1|t}$, the covariance matrix $P_{t+1|t}$, the observed measurement output g_t and the noise of measurement model R.

First of all, we transform the sample points for state quaternion to the sigma points for measurement. Here we need the measurement model $\mathcal{Z}_i = H(\mathcal{X}_i, 0)$ where also omit the noise and add it later:

$$[0, \mathcal{Z}_i] = \mathcal{X}_i\oplus g\oplus\mathcal{X}_i^{-1}\,,\ i = 0, 1, 2...2n \tag{11}$$

here g is the gravity in quaternion form $g = (0, 0, 0, -1)$. And we just pick the last three values in the quaternion output to form $\mathcal{Z}_i$

Notice that $\mathcal{Z}_i$ is a 3-by-1 vector and can just do the mean and covariance operation as any other vectors. So the mean and covariance is given below.

$$z_{t+1} = \sum_{i=0}^{2n}\alpha_{m,i}\mathcal{Z}_i \tag{12}$$

$$P_{zz} = \sum_{i=0}^{2n}\alpha_{c,i}(\mathcal{Z}_i - z_{t+1})(\mathcal{Z}_i - z_{t+1})^T \tag{13}$$

And by considering the residual error $e_{v,i}$, we can get the covariance between state quaternion and measurement.

$$P_{xz} = \sum_{i=0}^{2n}\alpha_{c,i}e_{v,i}(\mathcal{Z}_i - z_{t+1})^T \tag{14}$$

Then we add noise into consideration

$$P_{\nu\nu} = P_{zz} + R \tag{15}$$

And get Kalman gain by equation 16

$$K_t = P_{xz} P_{vv}^{-1} \tag{16}$$

Next we derive the bias by comparing sensor data to estimated value

$$\nu_t = g_t - z_{t+1} \tag{17}$$

Finally we will figure out the updated mean and covariance for the state quaternion

$$x_{t+1|t+1} = x_{t+1|t} - \exp([0, \frac{1}{2} K_t \nu_t]) \tag{18}$$

$$P_{t+1|t+1} = P_{t+1|t} - K_t P_{\nu\nu} K_t^T \tag{19}$$

B. Panorama Reconstruction

Panorama reconstruction is to first generate 3D RGB points given images and corresponding pose of camera (also needs assumptions for depth), then project it to one certain plane as panorama image.

Our main idea is just do a bunch of coordinate transformation and scaling. First to transform image-coordinate to sphere-coordinate and then to cartisian-coordinate. Next do the rotation operation to get the world frame cartisian-coordinate from robot frame cartisian-coordinate, based on camera orientation calibrated from the UKF part. After that, we turn cartisian-coordinate to sphere coordinate, and to cylinder coordinate, at last split the cylinder from its side surface to generate the panorama image. The first part and last part are related to canvas size so we need the scaling process.

It is time-consuming if we deal with the image pixel-wise. The better and faster way is to operate the whole image together. Though we implemented matrix-wise image-opers in the code, here for simplicity, we still just select one pixel in one image as an example to illustrate all the transformation.

So the input is the image-coordinate (i, j) of this exact pixel from the k-th image, with RGB value as $I_{k,i,j} = (I_{k,i,j,r}, I_{k,i,j,g}, I_{k,i,j,b})$, the depth of this pixel is 1 (as we assume all the points in real world are 1m from camera) the width and height of the camera plane as w and h, the width and height of the canvas plane as W and H, the range of view for camera horizontally and vertically as r_h, r_w, and the robot pose $u = (u_x, u_y, u_z)$. The output is the image-coordinate of this pixel on panorama canvas and its color(certainly this is also $(I_{k,i,j,r}, I_{k,i,j,g}, I_{k,i,j,b})$).

First we transform image-coordinate (i, j) to sphere-coordinate(ρ, θ, ϕ)

$$\begin{cases} \rho = 1 \\ \theta = \pi - (0.5h - i) \cdot r_h / h \\ \phi = (j - 0.5w) \cdot r_w / w \end{cases} \tag{20}$$

Then we transform sphere-coordinate (ρ, θ, ϕ) to cartisian-coordinate (x_r, y_r, z_r)

$$\begin{cases} x = \rho \sin(\theta) \cos(\phi) \\ y = \rho \sin(\theta) \sin(\phi) \\ z = \rho \cos(\theta) \end{cases} \tag{21}$$

Next we do the transformation from robot frame$X_r = (x_r, y_r, z_r)$ to world frame$X_w = (x_w, y_w, z_w)$ given the rotation matrix U from pose u

$$X_w = X_r U \tag{22}$$

By now, we have already got the 3D RGB data in real world. If we want to project this dataset to a image plane to generate a panorama image, we need to do the following steps.

To do that, we first transform the world frame cartisian coordinate (x_w, y_w, z_w) back to sphere coordinate $(\tilde{\rho}, \tilde{\theta}, \tilde{\phi})$

$$\begin{cases} \tilde{\rho} = 1 \\ \tilde{\theta} = arctan2(\sqrt{x_w^2 + y_w^2}, z_w) \\ \tilde{\phi} = arctan2(y_w, x_w) \end{cases} \tag{23}$$

Here $arctan2(.,.)$ is the signed $arctan()$ which range from $-\pi$ to π

Then we swift the sphere coordinate $(\tilde{\rho}, \tilde{\theta}, \tilde{\phi})$ to cylinder coordinate. That's just $(\tilde{\theta}, \tilde{\phi})$

Finally we do the shifting and scaling part to get canvas coordinate (i_c, j_c)

$$\begin{cases} i_c = (\tilde{\theta}) \cdot H / \pi \\ j_c = (\tilde{\phi} + \pi - 0.5 r_w) \cdot W / (2\pi) \end{cases} \tag{24}$$

After we adopt the same process to all the pixels in all the images we have, we can get a panorama image $\mathcal{I}_(i_c, j_c) = (I_{k,i,j,r}, I_{k,i,j,g}, I_{k,i,j,b})$.

IV. Results

For all the experiments afterwards, we use covariance as 0.001Σ for independant noises coming from motion model, measurement model and the initial covariance of the state quaternion.

As for the index of data, dataset 1,2,8,9 are training sets which means we have vicon data as ground truth. Dataset 10,11,12,13 are testing sets, which we don't have vicon data.

A. Trainset Performances

In training phase, we have 4 set of complete dataset with camera images, imu sensor data and vicon data(as ground truth). For each dataset, we first implemented integral method, only using motion model to see the performance when no calibration is added. Then we used only prediction step in UKF, discarding all the measurement data to see what will happen. Finally we used the whole UKF process for calibration and trying to get a better result. To make our experiment results more convincing, we also gave the plot comparing the predicted measurement value z_{t+1} and real measurement value.

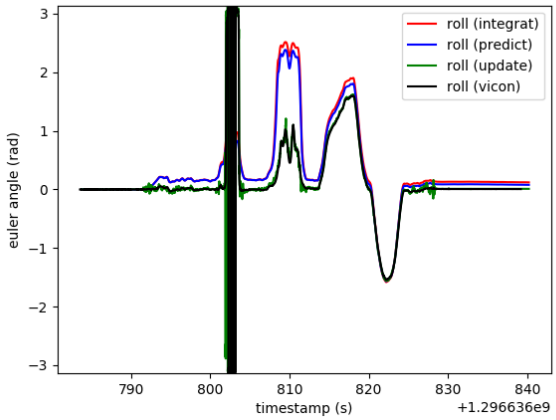

Fig. 1. Calibrated roll, dataset 1

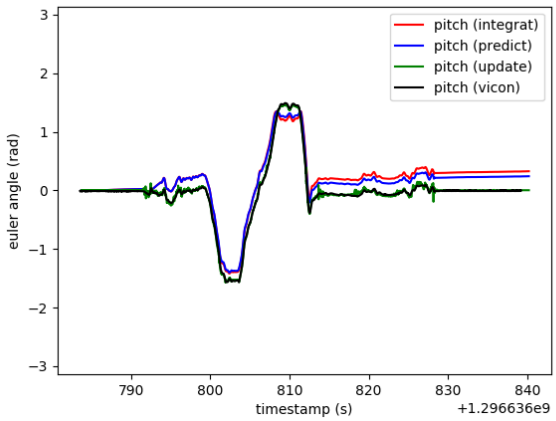

Fig. 2. Calibrated pitch, dataset 1

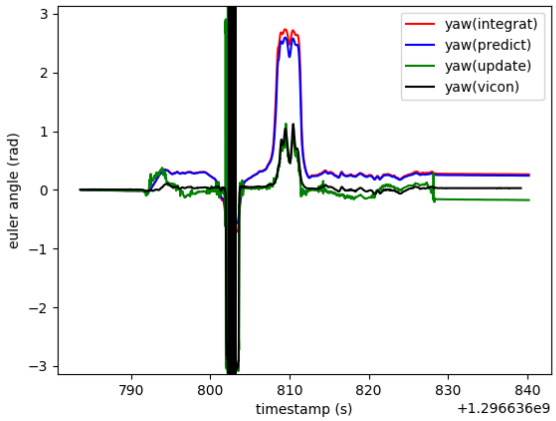

Fig. 3. Calibrated yaw, dataset 1

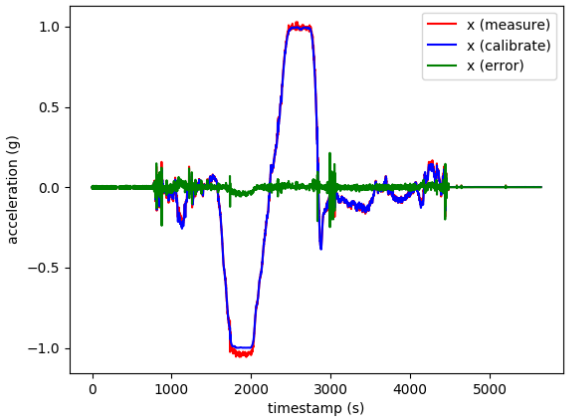

Fig. 4. Error in gx, dataset 1

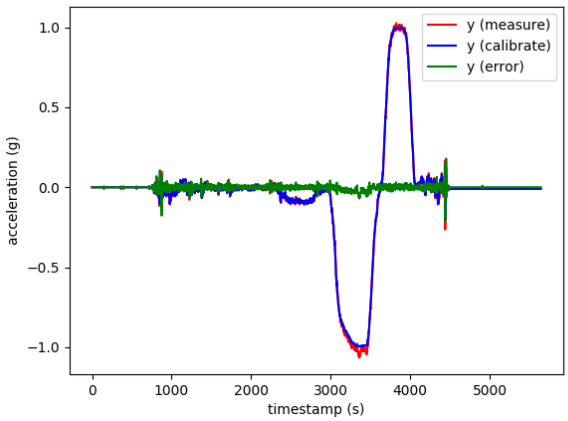

Fig. 5. Error in gy, dataset 1

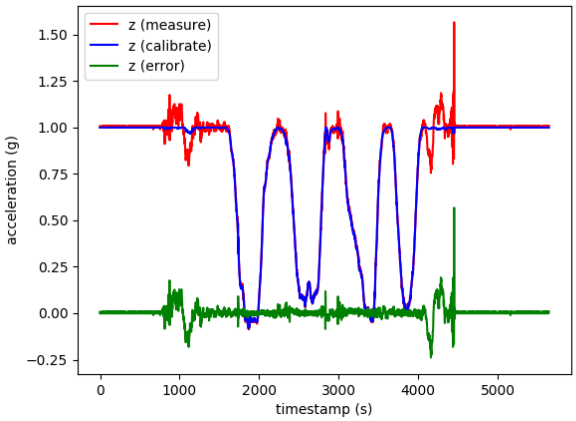

Fig. 6. Error in gz, dataset 1

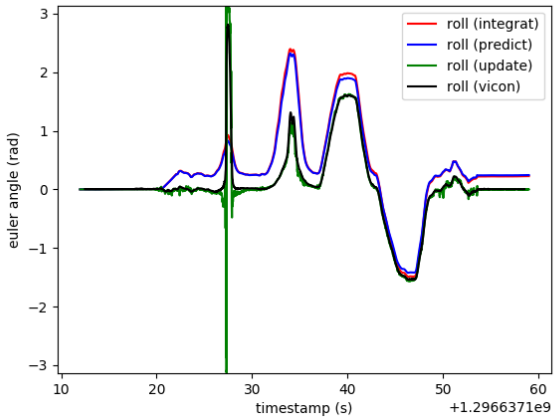

Fig. 7. Calibrated roll, dataset 2

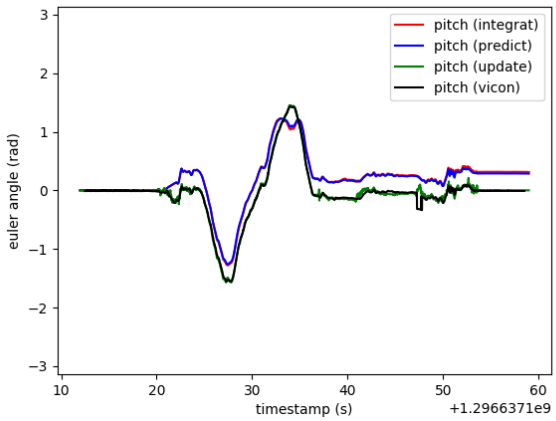

Fig. 8. Calibrated pitch, dataset 2

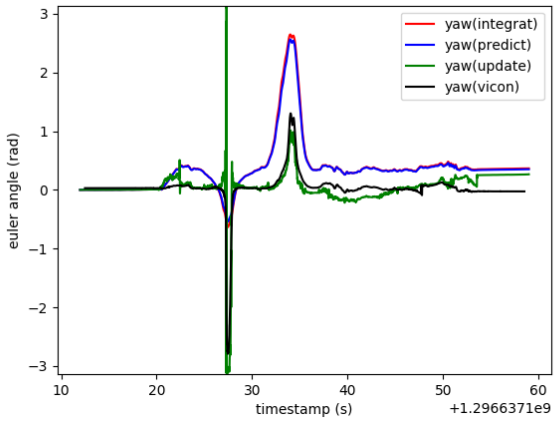

Fig. 9. Calibrated yaw, dataset 2

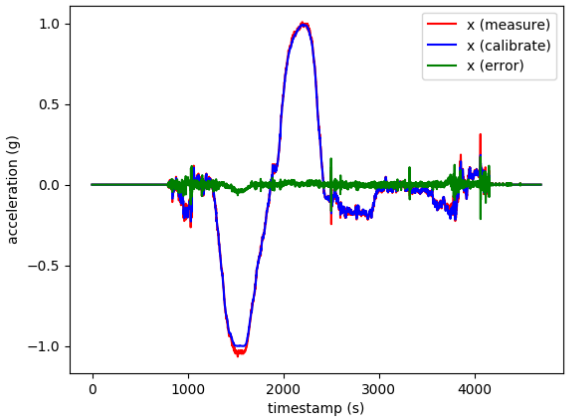

Fig. 10. Error in gx, dataset 2

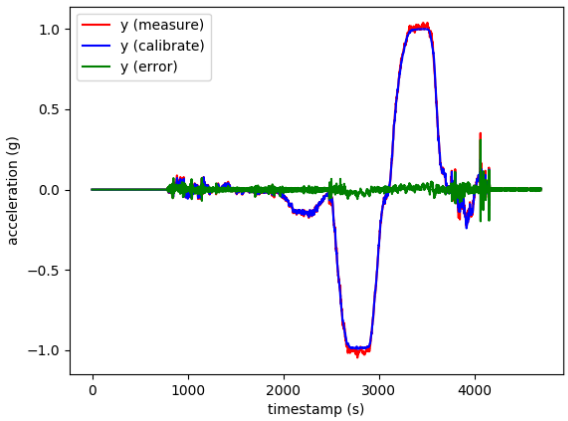

Fig. 11. Error in gy, dataset 2

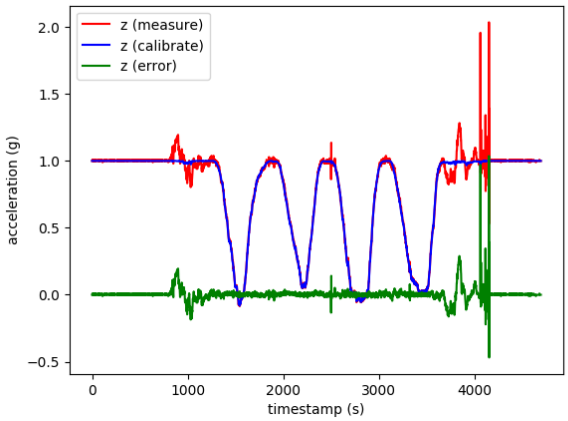

Fig. 12. Error in gz, dataset 2

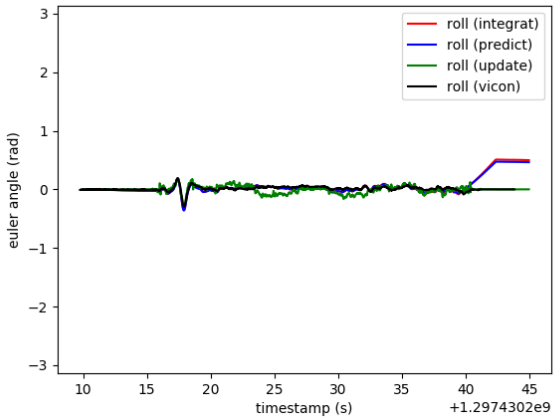

Fig. 13. Calibrated roll, dataset 8

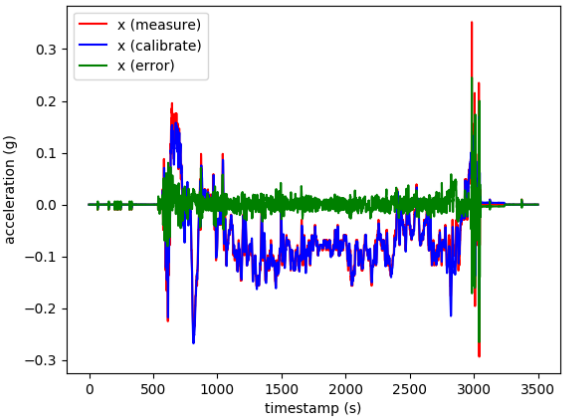

Fig. 16. Error in gx, dataset 8

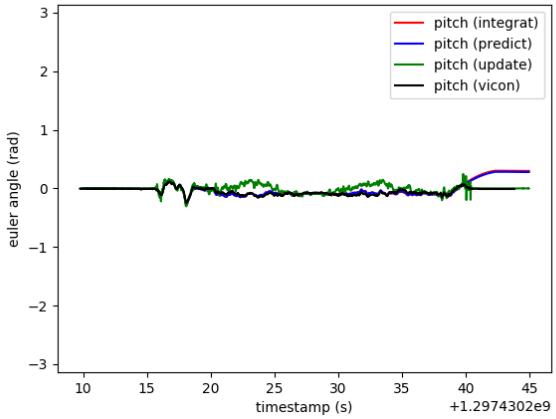

Fig. 14. Calibrated pitch, dataset 8

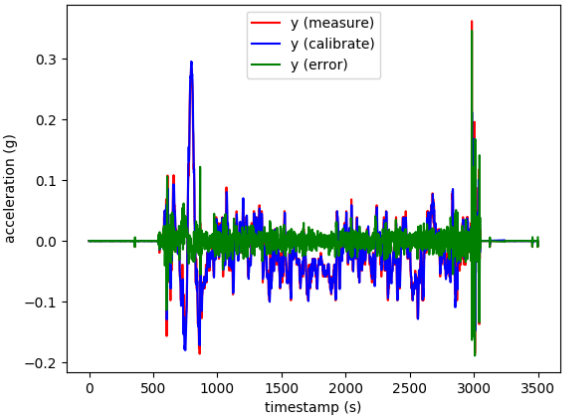

Fig. 17. Error in gy, dataset 8

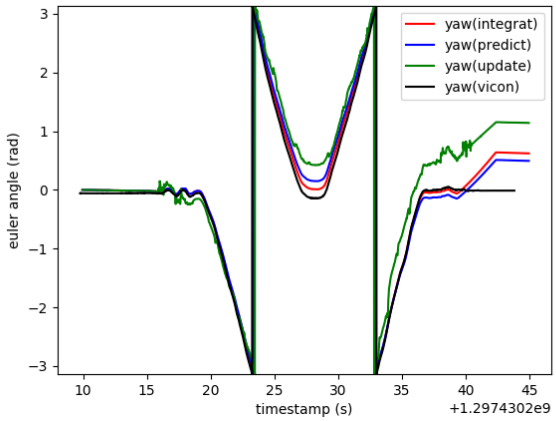

Fig. 15. Calibrated yaw, dataset 8

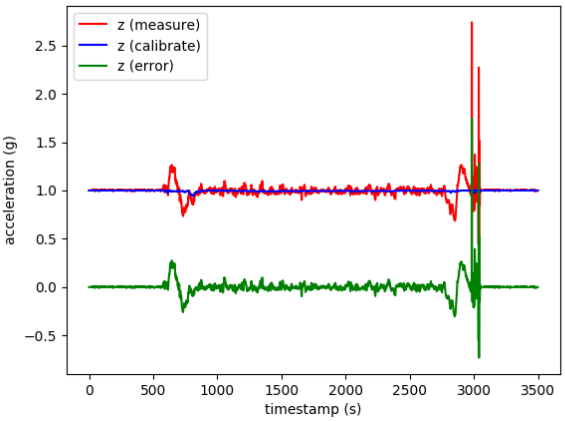

Fig. 18. Error in gz, dataset 8

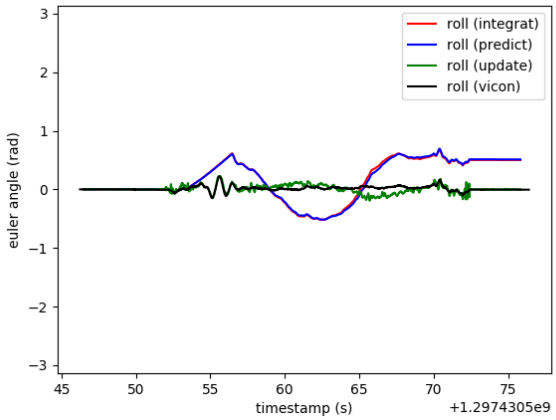

Fig. 19. Calibrated roll, dataset 9

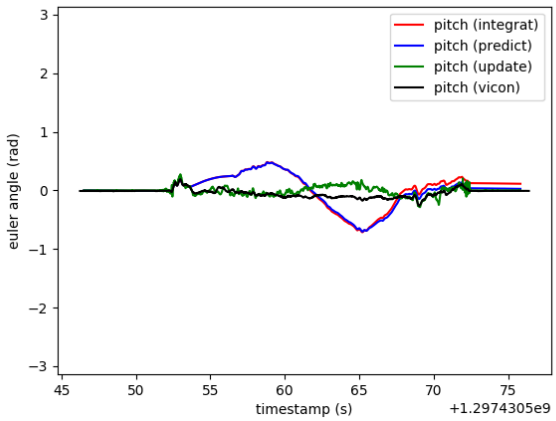

Fig. 20. Calibrated pitch, dataset 9

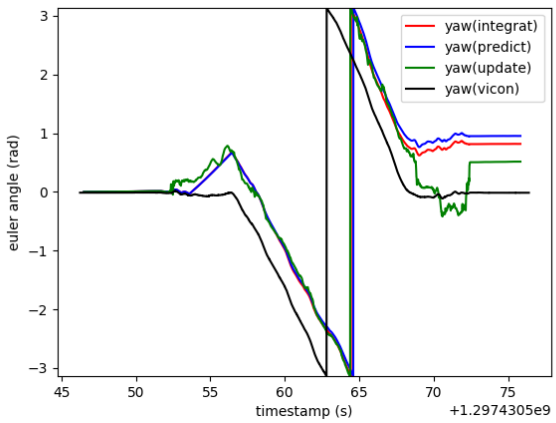

Fig. 21. Calibrated yaw, dataset 9

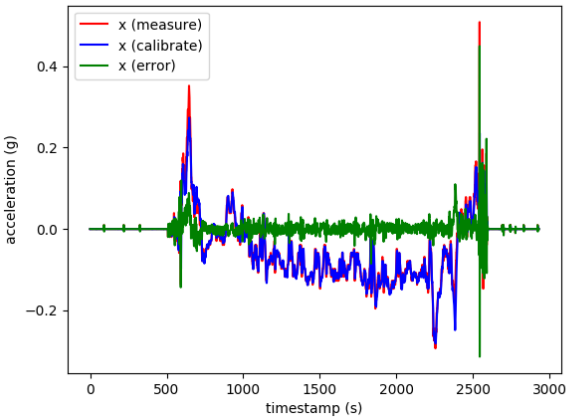

Fig. 22. Error in gx, dataset 9

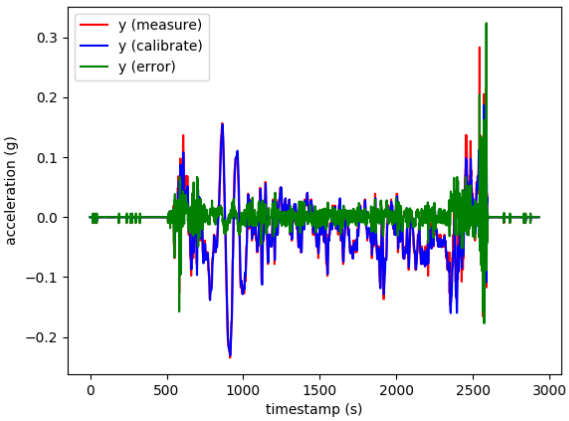

Fig. 23. Error in gy, dataset 9

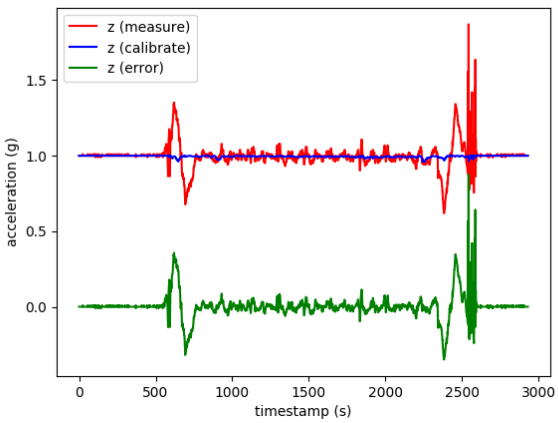

Fig. 24. Error in gz, dataset 9

From those plots can we see that, integrated data can basically match the trends of the ground truth, but can not catch the ground truth well in many cases (especially at local maximum and local minimum points). And predicted data acts mostly the same (if we enlarge the covariance then maybe the predicted data may not act as well as intergtate ones). But the most precised tracking of the ground truth comes to the updated data, that's the outcome of UKF calibration. It performs quite well in most of the scenarios. Big errors come to the "yaw" part and that may be caused by fierce rotation moves around the z-axis. Also we just use gravity as indicator. So when we just rotating through z-axis, with x-axis and y-axis having no acceleration, then we cannot estimate the yaw angle precisely because we lack informations in other two dimensions.

From the bias of estimated gravity direction and the measured gravity direction, we can also see why UKF performs well. In many cases, the error curve is just fluctuate slightly around zero, and that indicates that our UKF model can accurately estimate what the next measurement data could be, and make our estimation of orientation more certain. Even sometimes when big bias show up, it is only because the measurement data is absurd (i.e, linear acceleration shouldn't be larger than g) for commen sense (as Fig18. Error in gz, dataset 8 and Fig24. Error in gz, dataset 9).

B. Testset Performances

In testing phase, we have 4 set of complete dataset with camera images, imu sensor data. We don't know the vicon data so can only give the plot of the trends of intergrated data, predicted data and updated data. To make our experiment results more convincing, we also gave the plot comparing the predicted measurement value z_{t+1} and real measurement value.

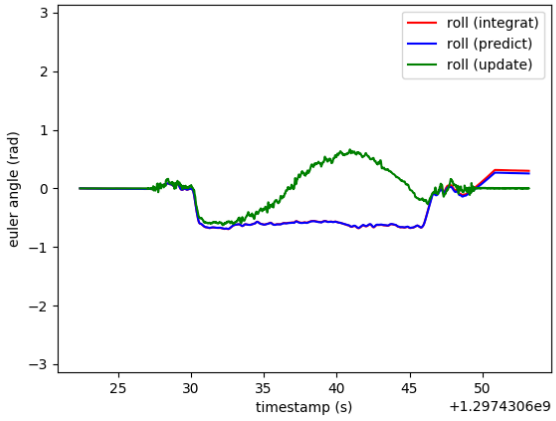

Fig. 25. Calibrated roll, dataset 10

Basically like the trainset part, the UKF updated data seems stable only in roll and pitch parts. When comes to estimation of yaw, the trends seems contains certain bias (though we don't

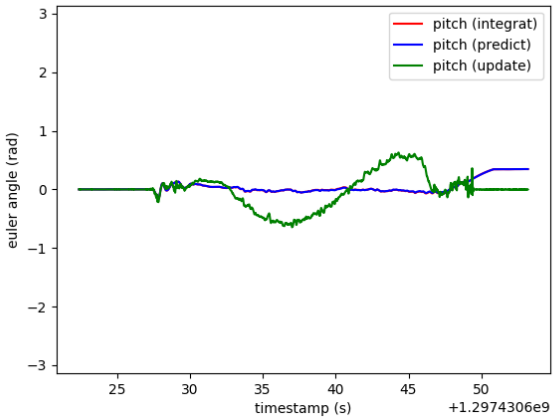

Fig. 26. Calibrated pitch, dataset 10

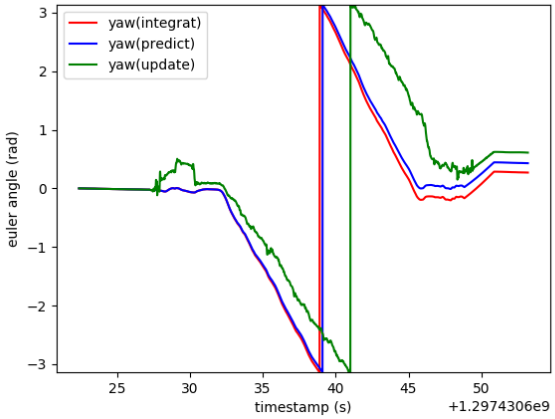

Fig. 27. Calibrated yaw, dataset 10

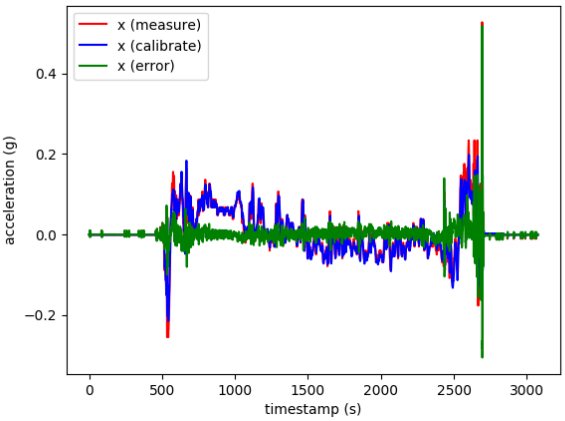

Fig. 28. Error in gx, dataset 10

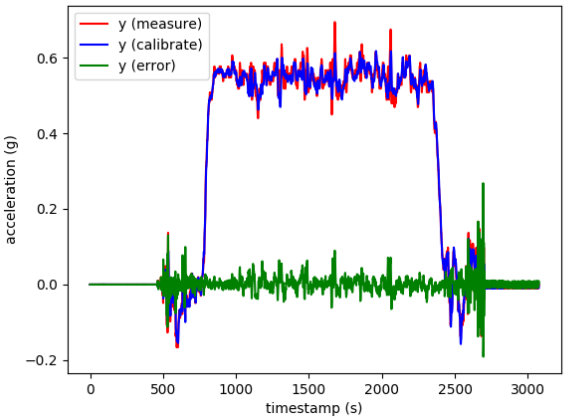

Fig. 29. Error in gy, dataset 10

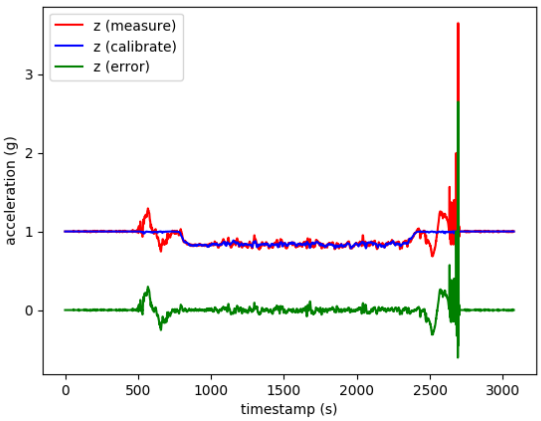

Fig. 30. Error in gz, dataset 10

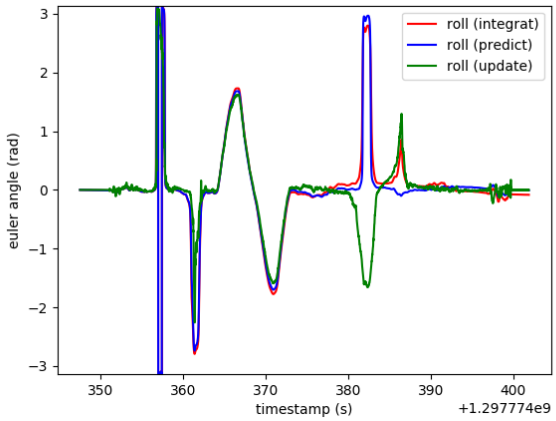

Fig. 31. Calibrated roll, dataset 11

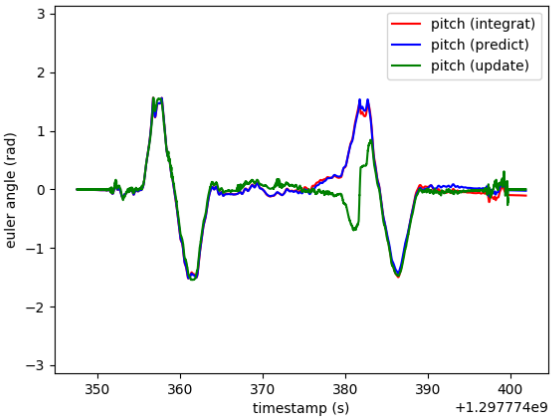

Fig. 32. Calibrated pitch, dataset 11

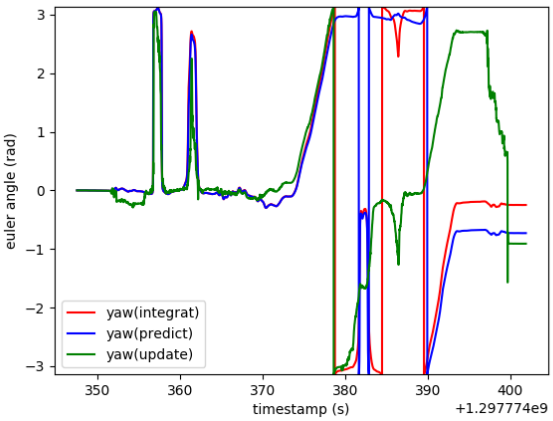

Fig. 33. Calibrated yaw, dataset 11

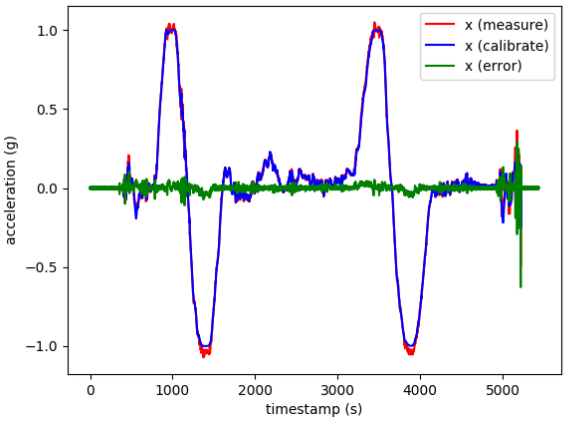

Fig. 34. Error in gx, dataset 11

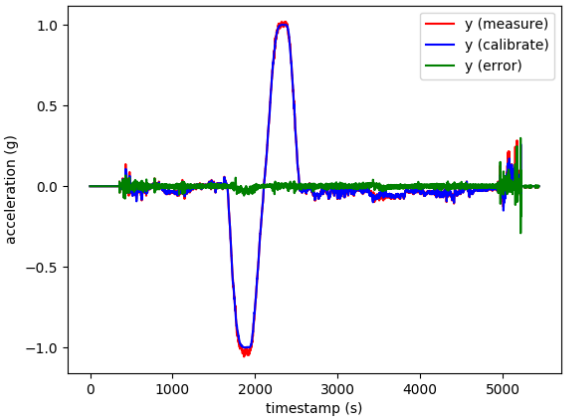

Fig. 35. Error in gy, dataset 11

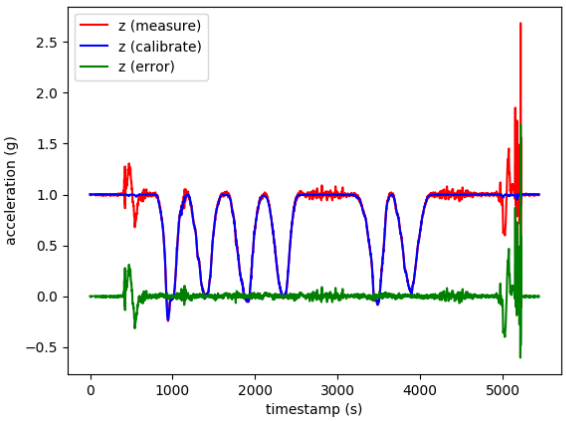

Fig. 36. Error in gz, dataset 11

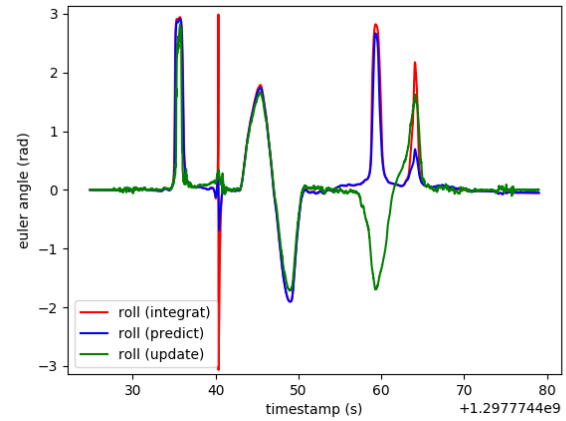

Fig. 37. Calibrated roll, dataset 12

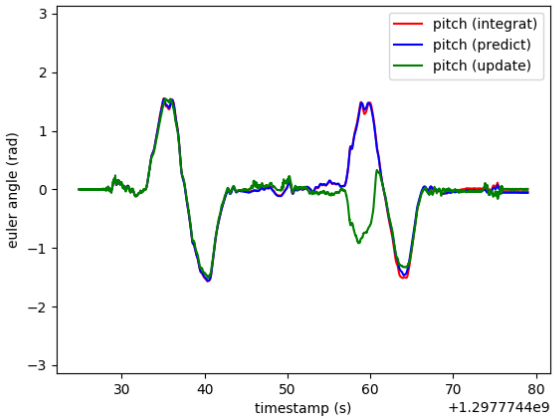

Fig. 38. Calibrated pitch, dataset 12

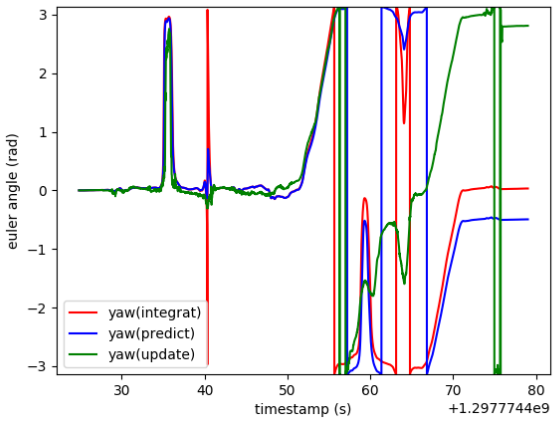

Fig. 39. Calibrated yaw, dataset 12

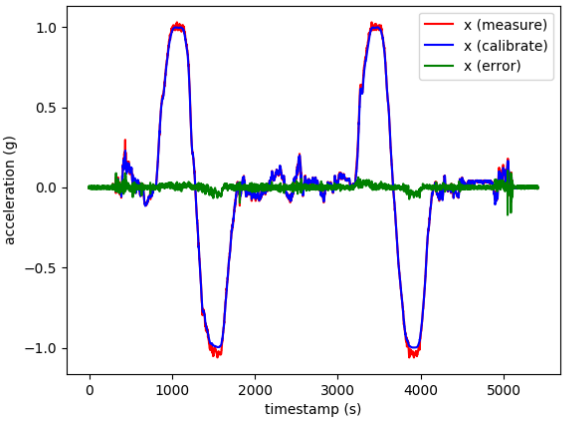

Fig. 40. Error in gx, dataset 12

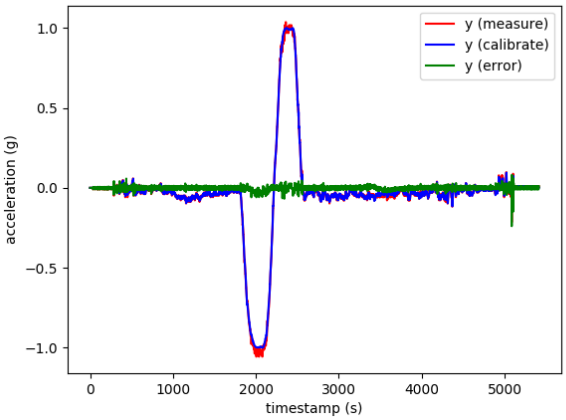

Fig. 41. Error in gy, dataset 12

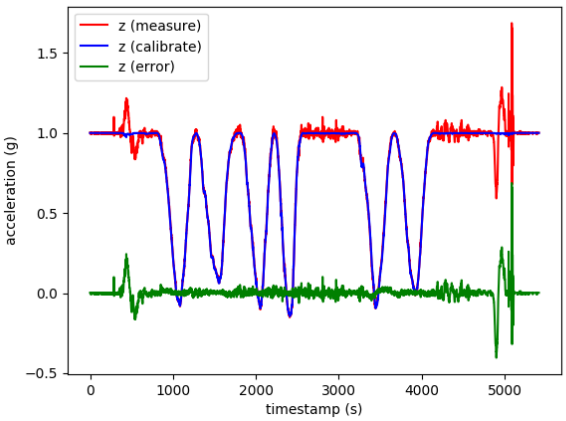

Fig. 42. Error in gz, dataset 12

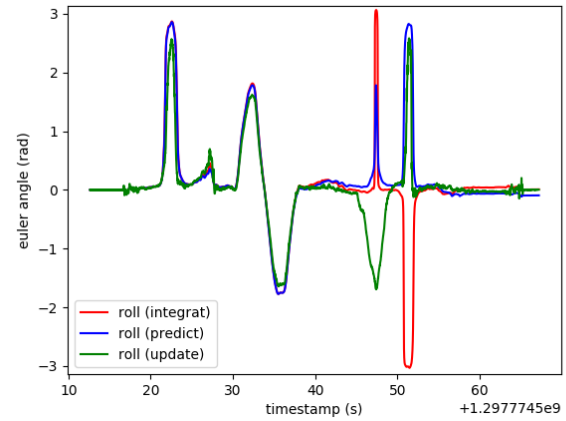

Fig. 43. Calibrated roll, dataset 13

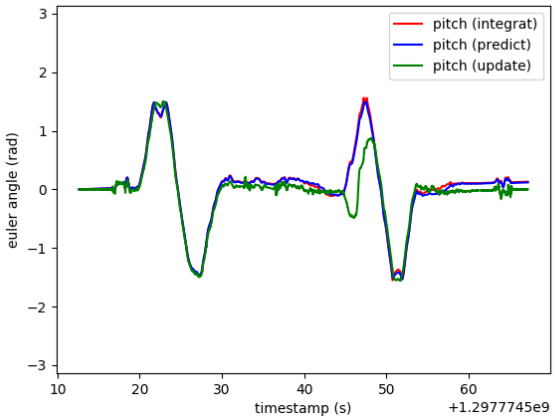

Fig. 44. Calibrated pitch, dataset 13

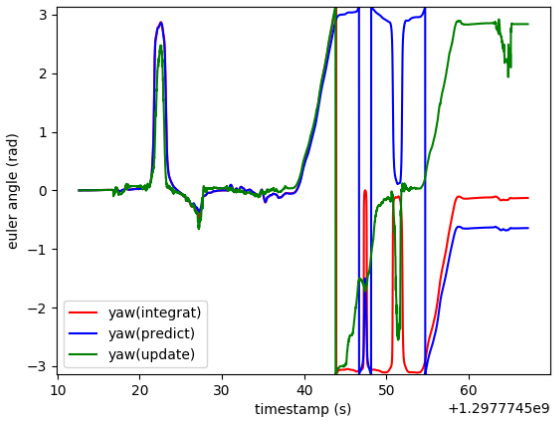

Fig. 45. Calibrated yaw, dataset 13

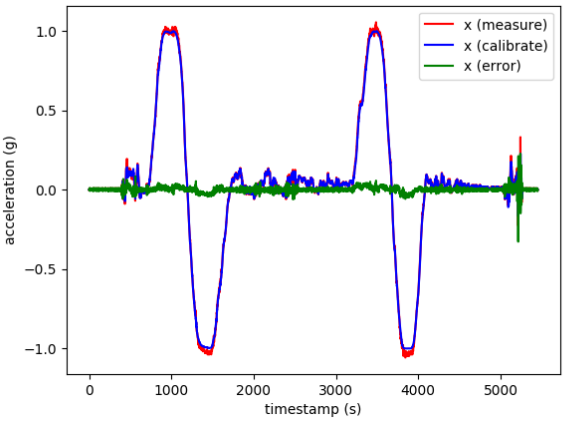

Fig. 46. Error in gx, dataset 13

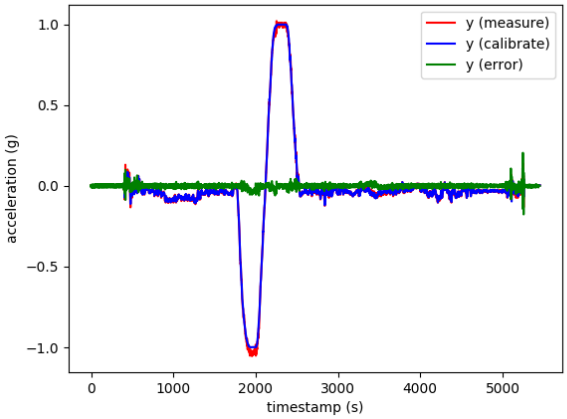

Fig. 47. Error in gy, dataset 13

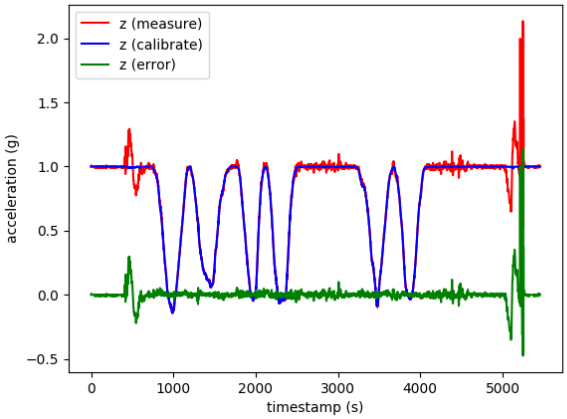

Fig. 48. Error in gz, testset 13

know the ground truth). And as for acceleration estimation in update process, as usual, the UKF estimation works fine except when the measurement data is unnaturally becoming too big.

C. Panoramic Results

1) Trainset Panoramic: Here we generated the panoramic images sourcing from both calibrated imu data and vicon data (as ground truth) as belows.

In dataset 1, 2 and 8, the panoramic image given by calibrated imu data and vicon data seems not have much difference. But when coming to dataset 9, the calibrated imu data gave twisted pattern. Actually this is the case when the camera only rotate through the z-axis in world frame. So we cannot estimate the yaw angle well. And that may lead to wrong panorama reconstruction result.

2) Testset Panoramic: In testing set, we generated 4 panoramic photos based on calibrated imu data.

The results are not that well. Dataset 2,3,4 have few camera images and also it seems that the position of camera is

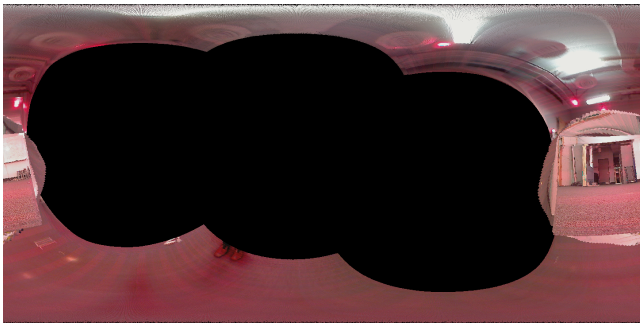

Fig. 49. Panoramic image from ukf, dataset 1

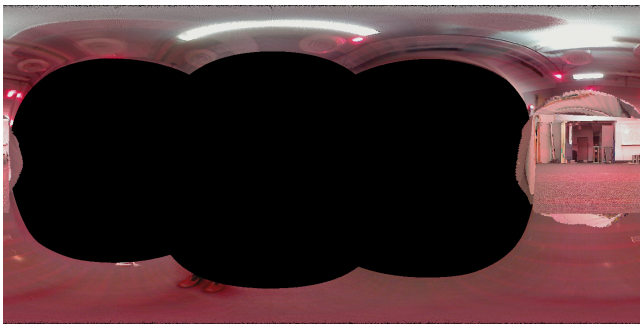

Fig. 50. Panoramic image from vicon, dataset 1

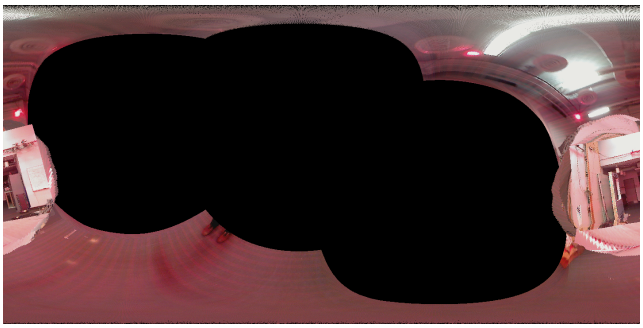

Fig. 51. Panoramic image from ukf, dataset 2

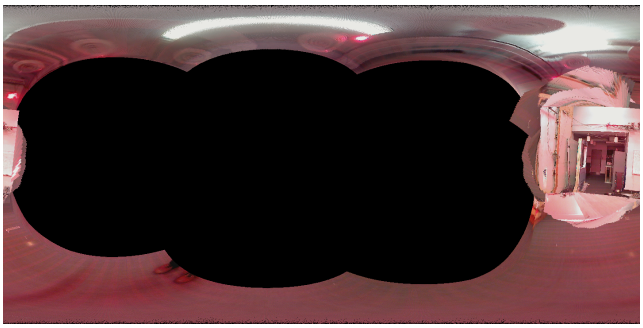

Fig. 52. Panoramic image from vicon, dataset 2

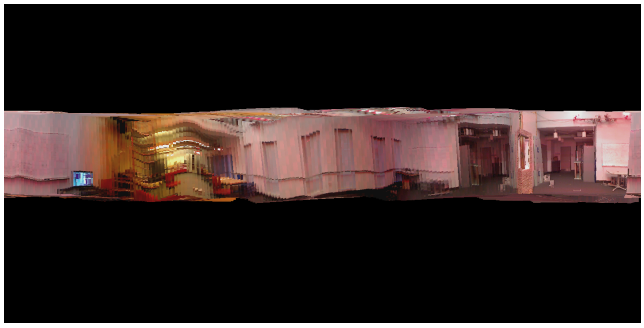

Fig. 53. Panoramic image from ukf, dataset 8

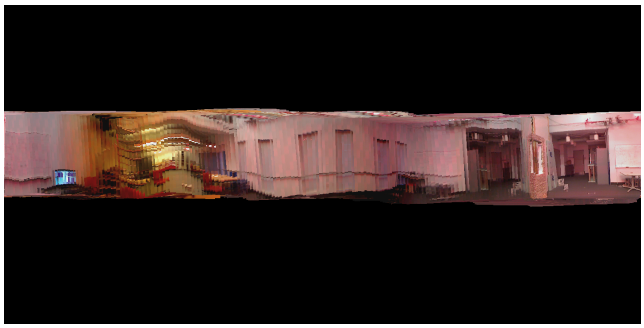

Fig. 54. Panoramic image from vicon, dataset 8

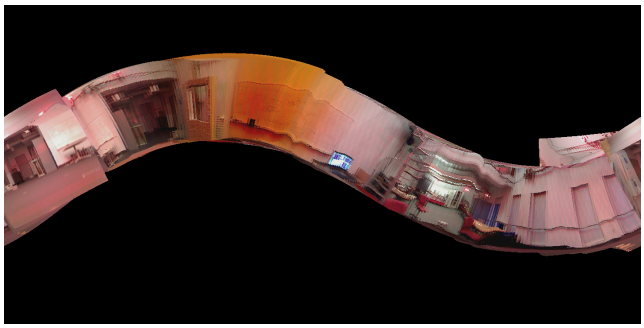

Fig. 55. Panoramic image from ukf, dataset 9

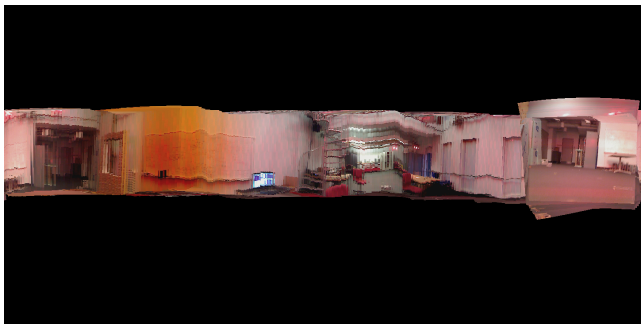

Fig. 56. Panoramic image from vicon, dataset 9

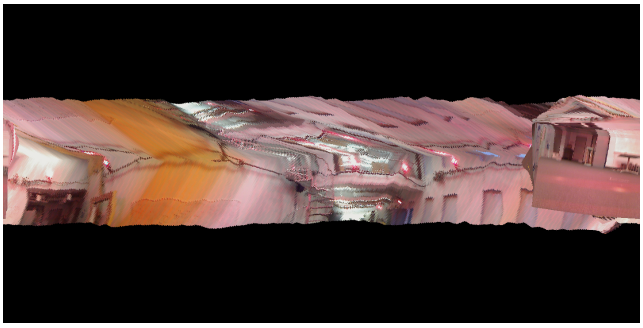

Fig. 57. Panoramic image from ukf, dataset 10

Fig. 58. Panoramic image from ukf, dataset 11

Fig. 59. Panoramic image from ukf, dataset 12

Fig. 60. Panoramic image from ukf, dataset 13

changing, which violate our initial assumption(camera only rotate). So it make sense to get this kind of results.

D. Conclusion

In all, we implemented the quaternion-based UKF to track the orientation of the camera and try to reconstruct a panorama image based on the calibrated result. In the experiments we can see the converge of the kalman filter results. Also the curve catches quite close with the ground truth vicon data. We presented fine result on training set. However, the panoramic part in testset is not that well and that may result from our measurement data cannot provide enough useful data in special movement cases(rotating through z-axis). Also the position of camera may be changed, which violated to our assumption that the camera can only rotate in fixed location. A suggestive way to refine the result is to fuse more sources of data (magentic field sensor, odometry data) into real application.

REFERENCES

[1] E. Kraft, A quaternion-based unscented Kalman filter for orientation tracking, Sixth International Conference of Information Fusion, 2003.

[2] J. Laviola, A comparison of unscented and extended Kalman filtering for estimating quaternion motion, Proceedings of the 2003 American Control Conference, 2003.