ESE 650, SPRING 2024

HOMEWORK 2

YUNPENG WANG [YUNPENGW@SEAS],

Solution 1 (Time spent: 5 hours)**.**

(a) I generate the data using the dynamics model in the handout. To generate Gaussian noise and the initial value of x, I use numpy.random.normal().

```
class EKF(object):
    def __init__(self):
        self.x_k_1 = lambda x_k, a, epsilon_k: a * x_k + epsilon_k
        self.y_k = lambda x_k, nu_k: np.sqrt(x_k ** 2 + 1) + nu_k
        self.epsilon_k = lambda: np.random.normal(loc=0, scale=1)
        self.nu_k = lambda: np.random.normal(loc=0, scale=1/2)
        self.x_0 = lambda: np.random.normal(loc=1, scale=2)

    def generate_KF_data(self, a=-1, steps=100):
        y_k_list = []
        x_k = self.x_0()
        for i in range(steps):
            x_k_1 = self.x_k_1(x_k, a, self.epsilon_k())
            y_k = self.y_k(x_k, self.nu_k())
            x_k = x_k_1
            y_k_list.append(y_k)
        return y_k_list
```

LISTING 1. Generate data

(b) To estimate a using EKF, I treat a as a part of the state. The dynamic model for a is,

$$a_{k+1} = a_k \tag{1}$$

The measurement model doesn't contain a. Therefore, the derivative of the measurement function over a is 0.

$$\frac{dy}{da} = 0 \tag{2}$$

Then I follow the EKF in the lecture notes to estimate a.

```
    def estimate_a(self, y_k_list: list):
        steps = len(y_k_list)
```

```
        mu_k_k = np.array([
            [1, -10]
        ])
        sigma_k_k = np.array([
            [2, 0],
            [0, 1]
        ])
        R = np.array([
            [1, 0],
            [0, 0]
        ])
        Q = 1/2
        mu_list = []
        sigma_list = []
        for i in range(steps - 1):
            # propagating the dynamics by one timestep
            # mu_k1_k and sigma_k1_k
            mu_x_k_k = mu_k_k[0, 0]
            mu_a_k_k = mu_k_k[0, 1]
            mu_k1_k = np.array([
                [mu_a_k_k * mu_x_k_k, mu_a_k_k]
            ])
            A = np.array([
                [mu_a_k_k, mu_x_k_k],
                [0, 1]
            ])
            sigma_k1_k = A @ sigma_k_k @ A.transpose() + R

            # incorporating the observation
            # kalman gain
            mu_x_k1_k = mu_k1_k[0, 0]
            C = np.array([
                [(mu_x_k1_k ** 2 + 1) ** (-1 / 2) * mu_x_k1_k, 0]
            ])
            K = sigma_k1_k @ C.transpose() * ((C @ sigma_k1_k @ C.transpose() +
Q) ** -1)

            # u_k1_k1 and sigma_k1_k1
            mu_k1_k1 = mu_k1_k + K.transpose() * (y_k_list[i + 1] - np.sqrt(
mu_x_k1_k ** 2 + 1))
            KC = K @ C
            sigma_k1_k1 = (np.identity(len(KC)) - KC) @ sigma_k1_k

```

```
            mu_list.append(mu_k1_k1[0, 1])
            sigma_list.append(sigma_k1_k1[1, 1])

            # update mu_k_k and sigma_k_k
            mu_k_k = mu_k1_k1
            sigma_k_k = sigma_k1_k1
        return mu_list, sigma_list
```

LISTING 2. Generate data

(c) I set the initial value of a to be -10 and the variance of a to be 1. The mean of a converges to the real value and the variance of a decreases quickly after the filter incorporates some observations.

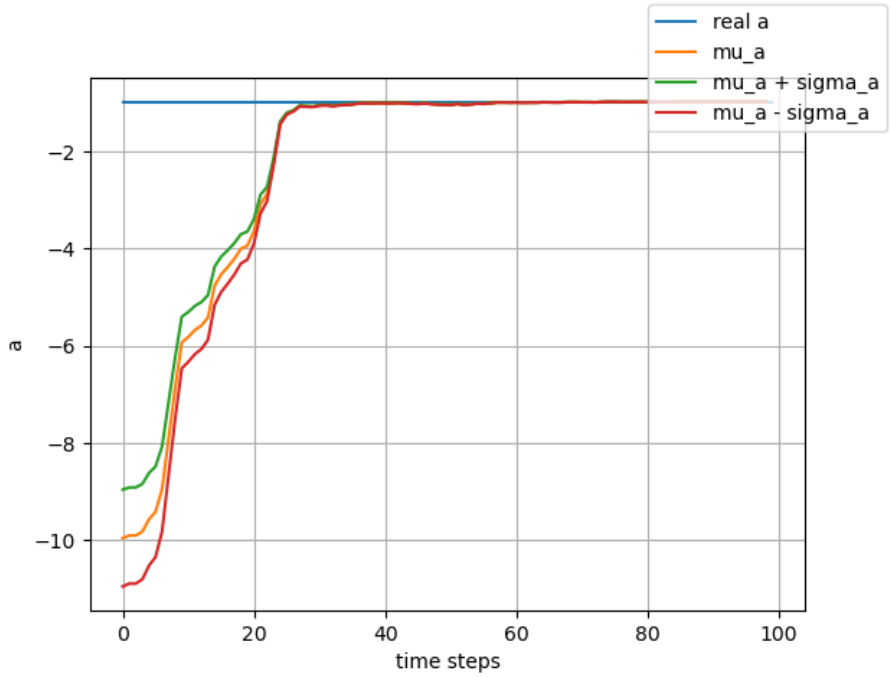

FIGURE 1. Estimate of a

Solution 2 (Time spent: 30 hour)**.**

(a)

```
for data_num in range(1, 4):
    imu = io.loadmat('imu/imuRaw'+str(data_num)+'.mat')
    print(imu)
    print(imu['vals'].shape)

    accel = imu['vals'][0:3,:] # accelerations in body frame
    print(accel)
    gyro = imu['vals'][3:6,:] # angular velocity
    print(gyro)
    T = np.shape(imu['ts'])[1]
    print(T)
    print(imu['ts'][0][0])
```

Python

```
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Wed Feb  2 12:53:30 2011', '__version__': '1.0', '__globals__': [], 'vals': array([[511, 511, 511, ..., 511, 511, 5
       [501, 501, 501, ..., 500, 500, 500],
       [605, 605, 606, ..., 605, 605, 606],
       [370, 369, 370, ..., 369, 369, 369],
       [374, 373, 374, ..., 373, 374, 373],
       [376, 376, 375, ..., 375, 376, 375]], dtype=uint16), 'ts': array([[1.29663678e+09, 1.29663678e+09, 1.29663678e+09, ...,
        1.29663684e+09, 1.29663684e+09, 1.29663684e+09]])}
(6, 5645)
[[511 511 511 ... 511 511 511]
 [501 501 501 ... 500 500 500]
 [605 605 606 ... 605 605 606]]
[[370 369 370 ... 369 369 369]
 [374 373 374 ... 373 374 373]
 [376 376 375 ... 375 376 375]]
5645
1296636783.735697
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Wed Feb  2 12:54:21 2011', '__version__': '1.0', '__globals__': [], 'ts': array([[1.29663711e+09, 1.29663711e+09, 1
       1.29663716e+09, 1.29663716e+09, 1.29663716e+09]]), 'vals': array([[511, 511, 511, ..., 511, 511, 511],
       [500, 500, 500, ..., 500, 501, 500],
       [605, 605, 605, ..., 605, 605, 605],
       [369, 371, 369, ..., 369, 369, 371],
       [373, 374, 373, ..., 374, 373, 374],
       [376, 376, 375, ..., 375, 376, 374]], dtype=uint16)}
(6, 4698)
[[511 511 511 ... 511 511 511]
 [500 500 500 ... 500 501 500]
 [605 605 605 ... 605 605 605]]
[[369 371 369 ... 369 369 371]
 [373 374 373 ... 374 373 374]
 [376 376 375 ... 375 376 374]]
4698
1296637111.981875
```

FIGURE 2. Data Exploration

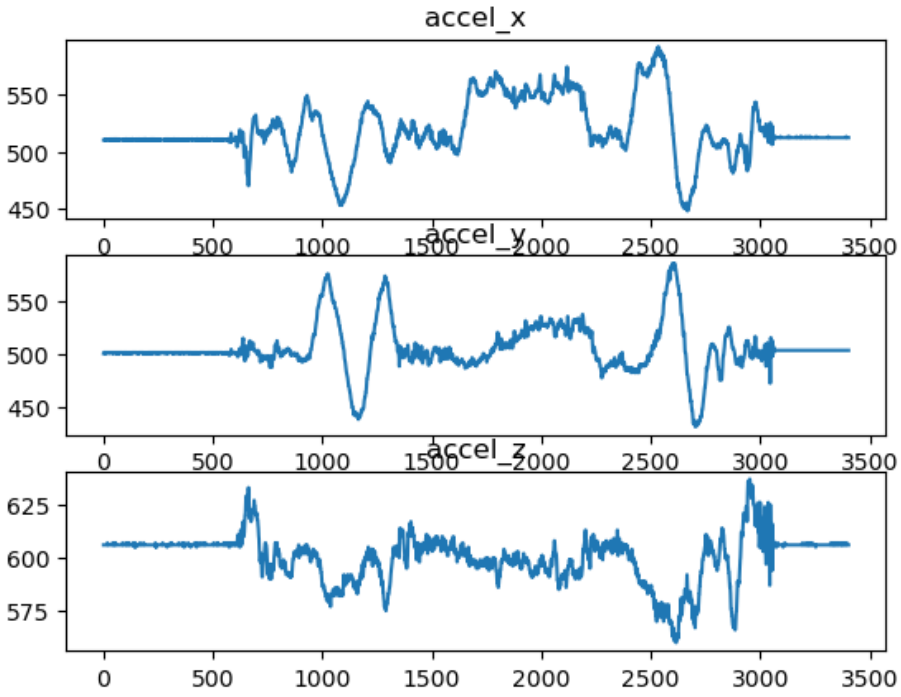

FIGURE 3. Accelerometer readings

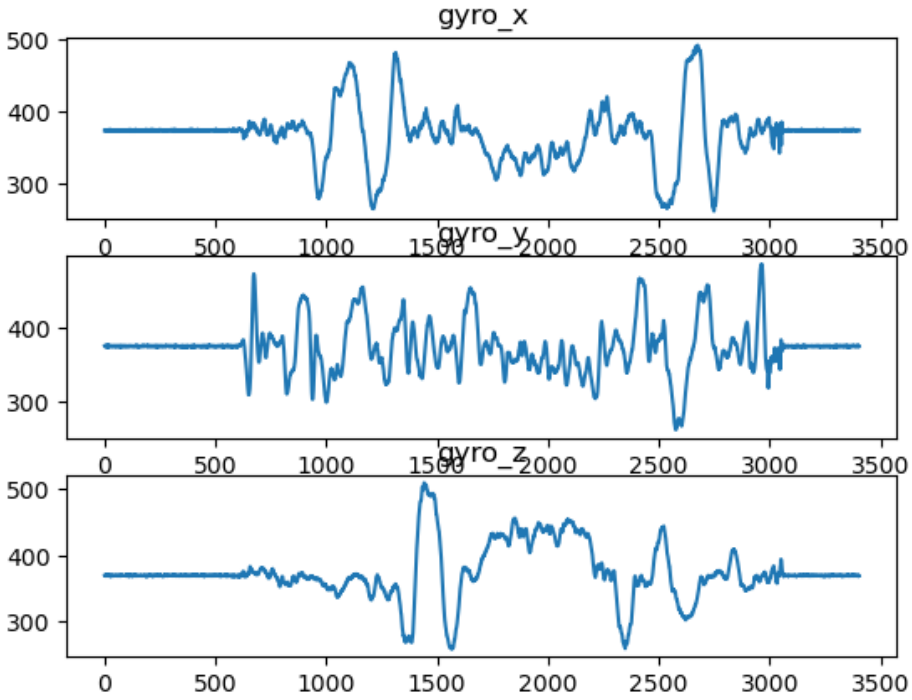

FIGURE 4. Gyroscope readings

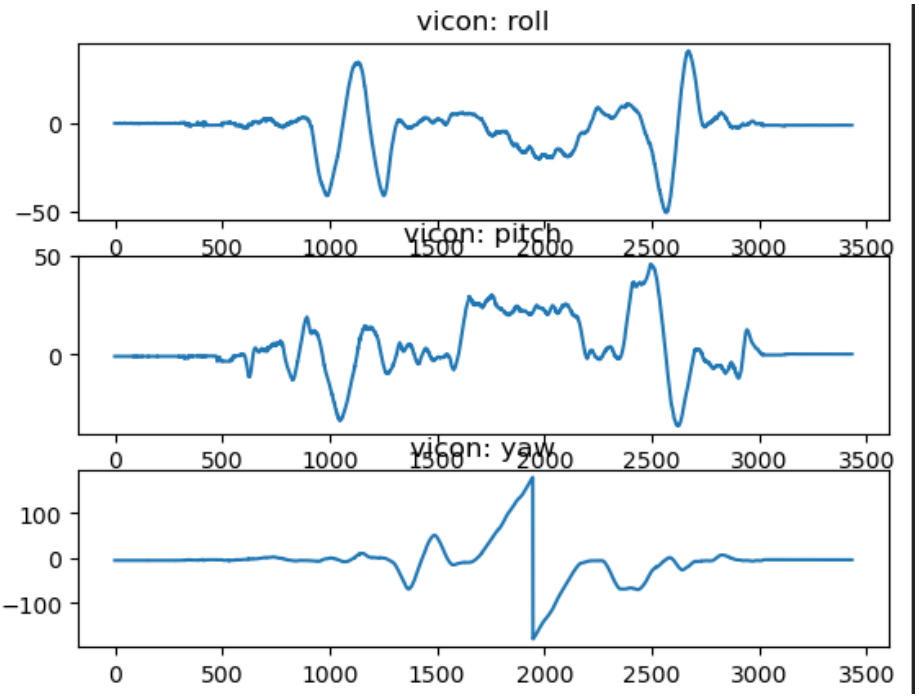

FIGURE 5. Rotations in Vicon

(b) I calibrated the sensors using the method talked about in the tutorial of this assignment. For the accelerometer, I assume that gravity is the only acceleration captured by the accelerometer. I transform the gravity in the world frame to the body frame with the rotation matrix from the Vicon dataset. Then I use linear regression optimization to estimate the bias and sensitivity. The sensitivity and bias along different axes are,

```
alpha_x, beta_x = -34.20366606015695, 510.96879722807637
alpha_y, beta_y = -34.25852812923489, 500.6043179840124
alpha_z, beta_z = -33.83737256121582, 502.45814226905196
```

LISTING 3. Generate data

The accelerometer calibration results are shown in Figure 6.

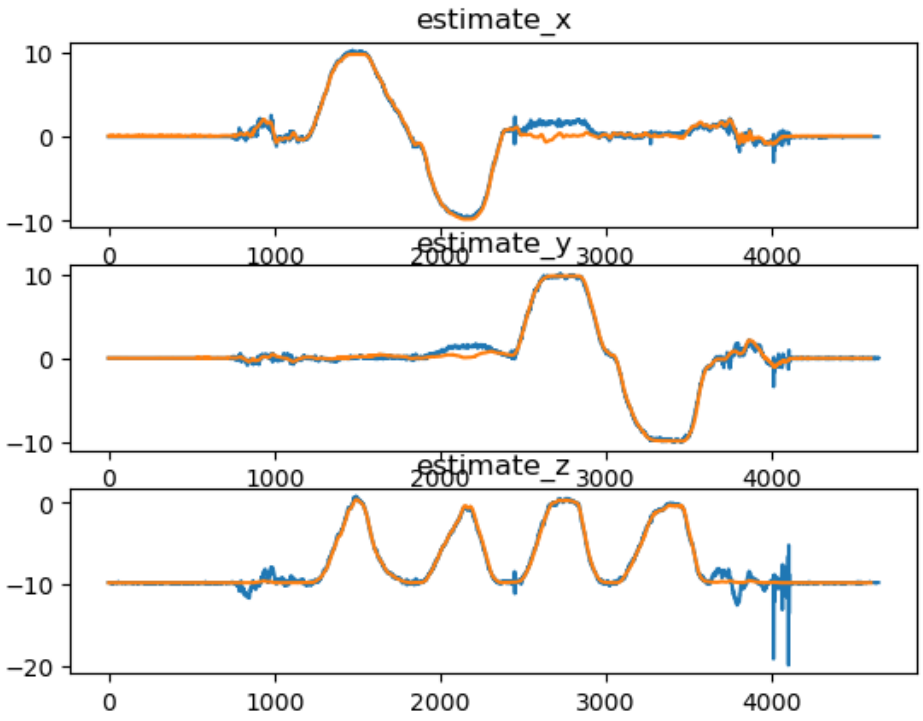

FIGURE 6. Accelerometer calibration

For the gyroscope, I calculate the real angular velocity from the Vicon dataset and estimate bias and sensitivity using linear regression. The sensitivity and bias are,

```
alpha_x, beta_x = 192.0673078010748, 373.7548883209933
alpha_y, beta_y = 180.25536722990273, 374.63586219640575
alpha_z, beta_z = 193.6674368840245, 369.5883024736752
```

LISTING 4. Generate data

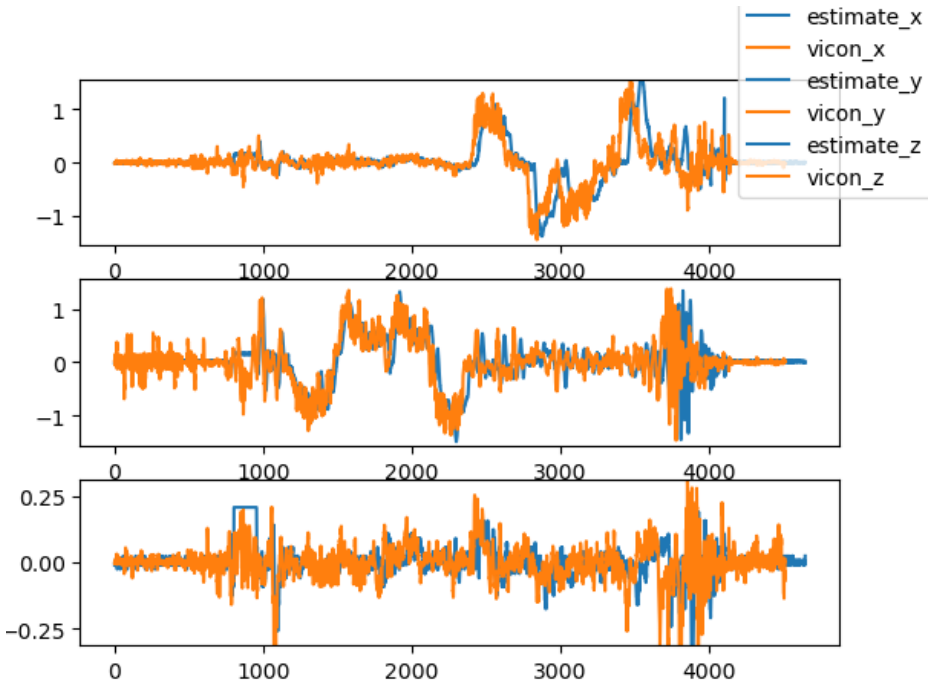

FIGURE 7. Gyroscope calibration

The gyroscope calibration results are shown in the Figure 7.

(c)

(d) The initial value of the covariance of the state, dynamics noise, and measurement noise I chose are as follows.

```
    sigma_0_0 = np.eye(6) * 0.1
    propagation_noise = np.eye(6) * 0.1
    observation_noise = np.eye(6) * 0.1
```

LISTING 5. Generate data

The approach I follow to implement UKF is the one documented in the EK's paper.

(e) The results of the orientation estimate in Euler angles are shown in Figure 8.

The results of the orientation estimate in quaternions are shown in Figure 9. Some parts of the figure show that there is a large difference between the estimate and the real values. However, the estimate and the real values are symmetric along $x = 0$ so the estimate of quaternion is just the other form of quaternion even if the estimate looks incorrect.

The results of the angular velocity estimate are shown in Figure 10.

The readings from the accelerometer and gyroscope are already shown in Figure 6 and Figure 7.

(f)

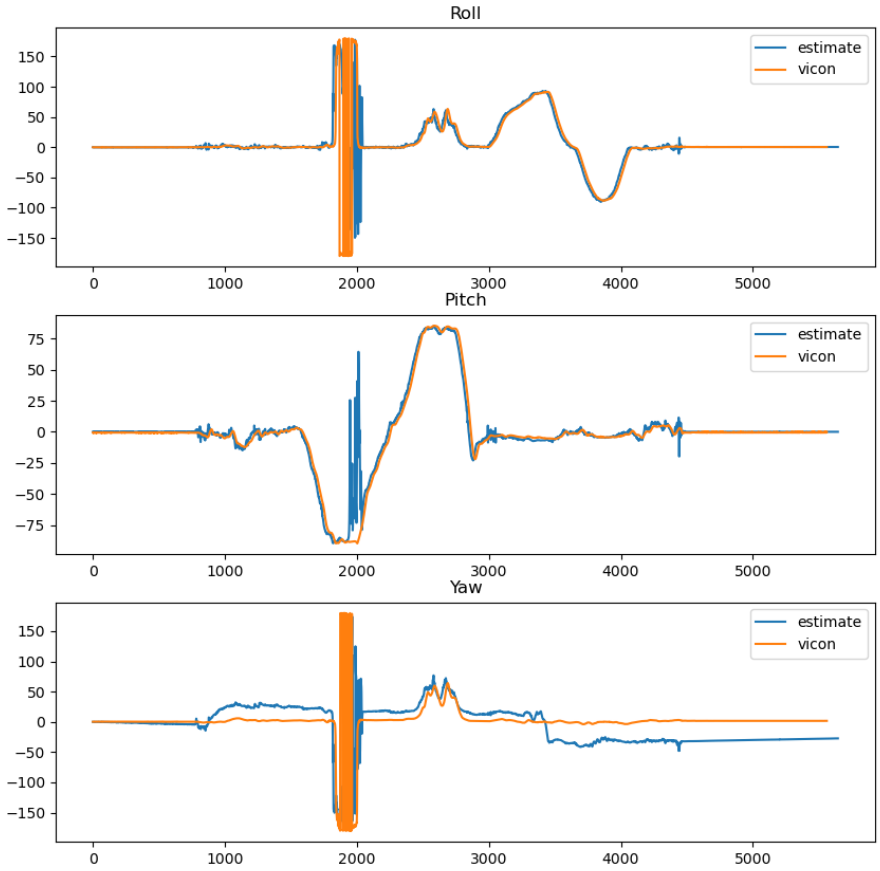

FIGURE 8. Orientation estimate in Euler angles

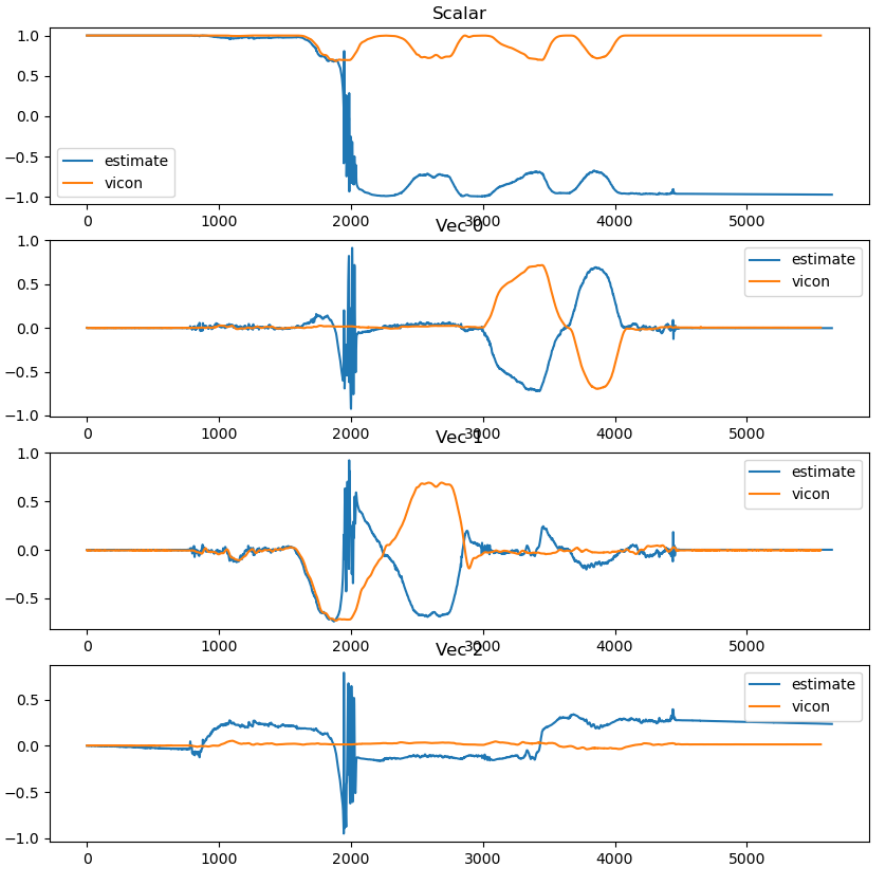

FIGURE 9. Orientation estimate in quaternion

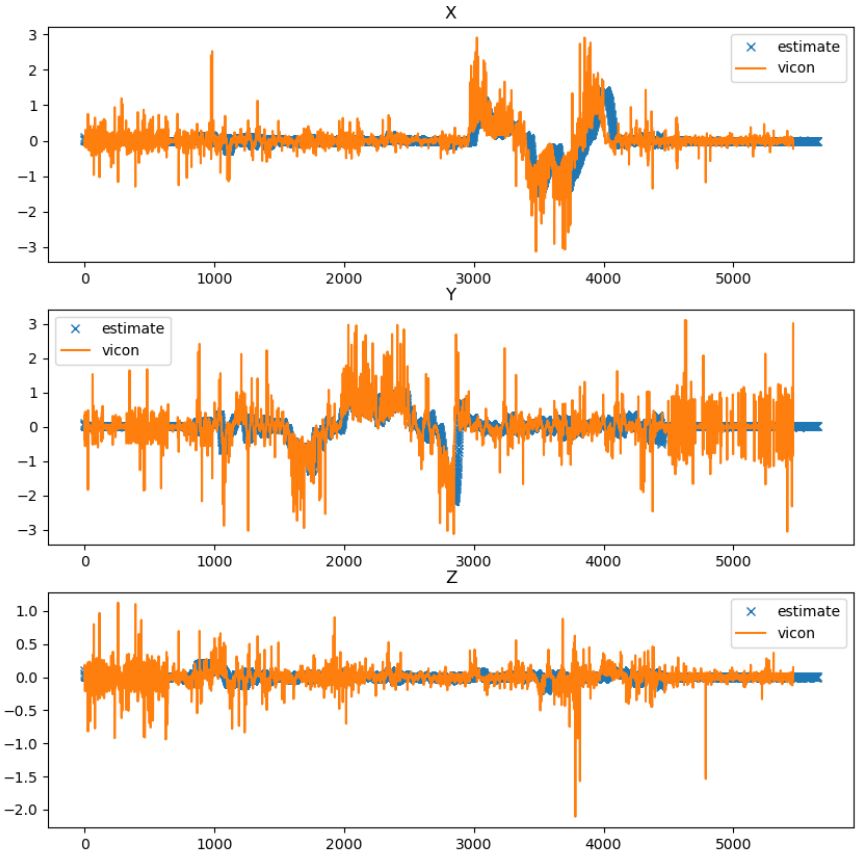

FIGURE 10. Angular velocity estimate