Homework #3 Report

[EECN30168] Self-Driving Cars 2022

Student ID: 311605004 **Name:** 劉子齊 **Date: 2022.10.19**

1. Introduction

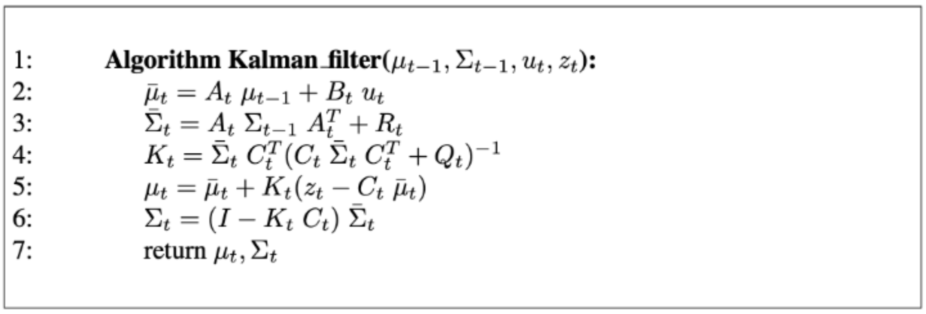
In homework 3, our goal is to understand Kalman Filter, and implement the Kalman filter through our own understandings. This may sound a bit simple, since we could have done this by simply call the Kalman filter from the FilterPy library. However, in this homework, we could only use NumPy and standard libraries of python.

For the implementation requirements, we have to implement the Kalman filter with the robot state [x, y, yaw]. Then we need to plot the output through the plotting function provided by the TAs, and analysis the result. I'll explain my implementation details in the following section.

2. Code Explanation

The following is the detail of my implementation of the predict and update function of the Kalman filter. Besides, I'll explain the design of the covariance matrices in the forth session.

When implementing my Kalman filter, I followed the algorithm provided by the TAs, which was as the following figure:



```
1:  Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):
2:      $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3:      $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4:      $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
6:      $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
7:      return $\mu_t, \Sigma_t$
```

```
def predict(self, u):
    self.x = np.dot(self.A, self.x) + np.dot(self.B, u)
    self.P = np.dot(np.dot(self.A, self.P), self.A.T) + self.Q
    # raise NotImplementedError

def update(self, z):

    S = self.R + np.dot(np.dot(self.H, self.P), self.H.T)
    K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
    y = z - np.dot(self.H, self.x)
    self.x = self.x + np.dot(K, y)

    I = np.eye(3)
    # self.P = np.dot(I - np.dot(K, self.H), self.P)
    self.P = np.dot(I - np.dot(K, self.H), self.P)
    return self.x, self.P
```

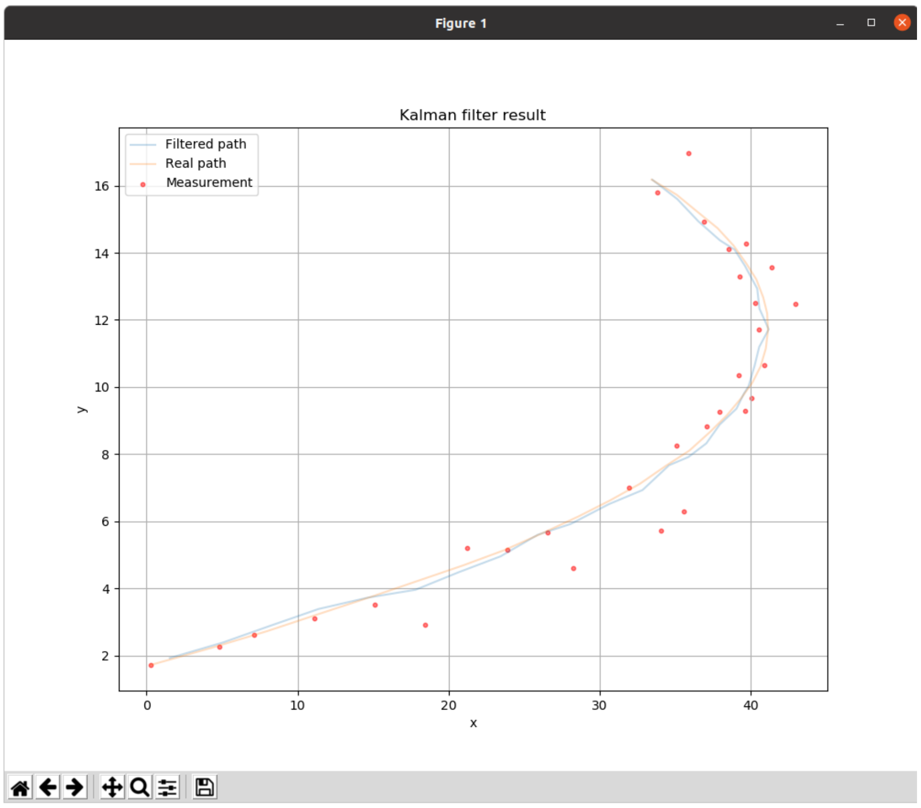
For the predict function, I first calculated the predicted state estimate x. I obtain the predicted state estimate x through adding the dot product of the transition matrix A and the state estimate x, and the dot product of the transition matrix B and the control vector u. Then I calculate the error matrix P through adding up the covariance of the state transition error Q with the dot product of the transition matrix A, the error matrix P and the transform of the transition matrix A.

For the update function, I first calculate the innovation covariance S, which can be found in the parentheses in the equation of the optimal Kalman gain K in the algorithm above. For the innovation covariance S, I added up the measurement error R with the dot product of the observation matrix H, the error matrix P, and the transform of the observation matrix H. Then I obtain the optimal Kalman gain K by calculating the dot product of the error matrix P, the transform of the observation matrix H, and the inverse of the innovation covariance S matrix.

Then I'll calculate the measurement post-fit residual y, which is the subtraction of the dot product of the observation matrix H and the state estimate x to the matrix z. With the measurement post-fit residual y, we can get the final state estimate x by adding the state estimate x itself with the dot product of the optimal Kalman gain K and the measurement post-fit residual y.

Lastly, for the error matrix P, we first initialize a 3x3 matrix I with np.eye(), which with ones on the diagonal and zeros elsewhere. Then we calculate the error matrix P by calculating the dot product of the subtraction of the dot product of the optimal Kalman gain K and the observation matrix H from the 3x3 matrix we just created and the error matrix P itself. Finally, we get the final error matrix P, and the following figure is the plotting result of the Kalman filter I implemented.

3. Result



4. Design of the Covariance Matrices (Q, R)

The following figure is how I designed my covariance matrices:

```
# State transition error covariance
self.Q = np.array([[0.08, 0.0, 0.0],
                   [0.0, 0.009, 0.0],
                   [0.0, 0.0, 1.0]])
```

```
# Measurement error
self.R = np.array([[0.5, 0.00],
                   [0.00, 0.7]])
```

For the state error covariance Q, I set the three values on the diagonal into 1.0 initially. First, I realized that in this case the third value, which is in (3, 3), doesn’t really affect the curve plotted in this case. Also, since the filtered path offsets from the real path more on the x–axis, besides, I consider the filter updates too much on the filtered path, which made the path to tangle greatly around the real path.

Hence, I started with adjusting the first value, which is in (1, 1). Then I found that as I lower the first value, the filtered path will fit more to the real path as it gets closer to the two endpoints of the curve. After getting the best value of the first value in (1, 1), which I considered to be "0.08", I started to adjust the second value, which is the value in (2, 2), and after lowering it all the way down to 0.009, I got the best result with the filtered path fit the most to the real path.

For the measurement error R, I also set the two values on the diagonal into 1.0 initially. Then I found the measurement points offset more on the y–axis from the real path, when I lower the two values, I always keep the second value on the (2, 2) slightly greater than the first value on the (1, 1) trying to average the difference on the offset. Finally, I set the two values correspondingly into 0.5 and 0.7.

5. Affection of the value of Q & R

Since the Kalman filter can be viewed as a combination of the distribution from which the state estimates, the value of the state error covariance Q provides a measurement of the width of the Gaussian distribution related to each noise state. As the distribution gets wider, the uncertainty found in the process model will also increases, which will make the update to have less influence on the particular state, and might eventually become with no influence.

For the measurement error R, I think the affection of the value is similar to the affection of the value of the state error covariance Q. As the value of the measurement error R gets greater, the update on the measurement point will be greater, and might update too much. On the other hand, if the value of the measurement error R is too small, will make the update to have less influence on the particular state, and might eventually become with no influence.