Report of HW3

> Student ID : A111137
>
> Name : 邱柏鎧

Introduction

This is the homework of implement the KF algorithm, the input contains displacement of robot and yaw change, that is [delta_x, delta_y, delta_yaw]. The state is setting as [x, y, yaw], however, we don't update the state of yaw.

In my works, sets the covariance of state transition error as 0 mean, 0.1 variance Gaussian noise, and sets the covariance of measurement error to 0 mean, 0.75 variance. The final result seems to be well even the measurement is extremely inaccurate sometimes.

Code Explaination

- Setting Covariance

  To fit the algorithm in the book, I change the definition of R as state transition error covariance, and the Q for measurement error.

  ```
  # State transition error covariance
  self.R = np.array([[0.1, 0, 0],
                     [0, 0.1, 0],
                     [0, 0, 0.1]])

  # Measurement error
  self.Q = np.array([[0.75, 0, 0],
                     [0, 0.75, 0 ],
                     [0, 0, 0.75]])
  ```

- Predict

  Because we do not update the value of yaw, so the covariance will become larger and larger, to prevent the value become inf, I set covariance of yaw always become a large enough value (the value means nothing).

  ```
  def predict(self, u):
      self.x = np.matmul(self.A,self.x) + np.matmul(self.B,u)
      self.P = np.matmul(np.matmul(self.A,self.P),np.transpose(self.A))
  + self.R
      ## Cause we don't update yaw, we need to prevent
      ## covariance of yaw become inf
      self.P[2,2] = 1.0
  ```

- Update

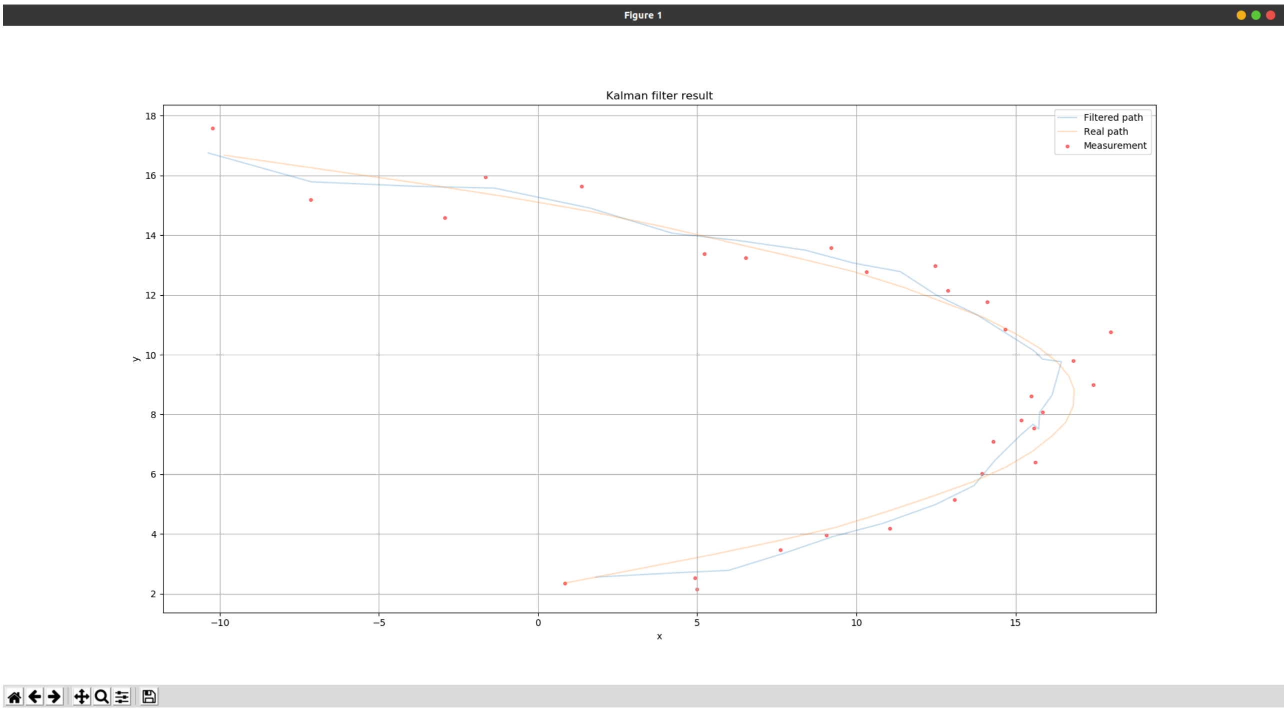
  My strategy is making the state always [x, y, yaw], even we don't need the state of yaw. I make z become 3*1 and H become 3*3 matrix. Other setting is same as the pseudo code of KF algorithm.

```
def update(self, z):
    ## Make z become 3*1 and H become 3*3 matrix
    im_z = np.transpose(np.append(z, 0))
    im_H = np.vstack([self.H, np.array([0, 0, 0])])
    ## Kalman Filter update part
    temp_sigma = np.matmul(np.matmul(im_H, self.P),
np.transpose(im_H)) + self.Q
    Kt = np.matmul(np.matmul(self.P, np.transpose(im_H)),
np.linalg.inv(temp_sigma))
    self.x = self.x + np.matmul(Kt,(im_z - np.matmul(im_H, self.x)))
    self.P = np.matmul((np.identity(3) - np.matmul(Kt, im_H)), self.P)

    if np.isnan(np.sum(self.x)) == True :
        raise ValueError

    return self.x, self.P
```

- Result



Design and Question

1. How you design the covariance matrices(Q, R)?

   Follow the definition, the 3*3 matrix of covariance should be

$$\begin{bmatrix} var(x) & cov(x,y) & cov(x,yaw) \\ cov(y,x) & var(y) & cov(y,yaw) \\ cov(yaw,x) & cov(yaw,y) & var(yaw) \end{bmatrix}$$

So the setting of R and Q will become

$$R = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.75 & 0 & 0 \\ 0 & 0.75 & 0 \\ 0 & 0 & 0.75 \end{bmatrix}$$

2. How will the value of Q and R affect the output of Kalman filter?

   Q (for measurement error) and **R (for state transition error)** represent the relability of measurement or state transition respectively. The larger covariance matrix is, the more we don't believe in. We can check this opinion by reverse Q and R, and the KF result will tend to follow the measurement result. The follow picture is the result of this thought.