Self-Driving Cars

309611087 洪得瑜

2022 10/20

1 Kalman Filter

- Introudction
- Briefly explain your code
- Include the ouput Image
- How you design the covariance matrices(Q, R)?
- How will the value of Q and R affect the output of Kalman filter?

1.1 Implement

1.1.1 Init[2]

1. x 為狀態矩陣包含 [x y yaw] 初始值設定為 [0 0 0]
2. A 為狀態轉換矩陣它將時間 t-1 的每個系統狀態參數的影響應用於時間 t 的系統狀態在這設定為 I
3. u_t 為控制參數如油門或者位置控制等
4. B 為輸入控制矩陣將 u_t 的控制參數影響狀態向量
5. P 為協方差矩陣 (Error matrx) 對角線為是狀態向量中對應相關方差, 非對角項為狀態向量之間的協方差, 在這初始設定為 I 彼此獨立, 下一時刻會更新新的協方矩陣
6. H 為狀態向量轉換到量測向量轉換矩陣在這設定 $\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}$, 因此對於 yaw 並不影響量測狀態
7. Q 矩陣為狀態協方矩陣和控制 noisy 有關, 影響 predict, 在這控制項為 3 項因此為 3X3 矩陣
8. R 矩陣為量測 noisy 協方矩陣, 量測結果為 x 及 y 因此為 2X2 矩陣

```
3 class KalmanFilter():
4     def __init__(self, x=0, y=0, yaw=0):
5         # State [x, y, yaw]
6         self.x = np.array([x, y, yaw])
7         # Transition matrix
8         self.A = np.identity(3)
9         self.B = np.identity(3)
10        # Error matrix
11        self.P = np.identity(3) * 1
12        # Observation matrix
13        self.H = np.array([[1, 0, 0],
14                           [0, 1, 0]])
15        # State transition error covariance
16        self.Q = np.array([[1,0,0],
17                           [0,1,0],
18                           [0,0,1]])
19        # Measurement error
20        self.R = np.array([[1,0],
21                           [0,1]])
22
23        self.I = np.eye(3)
24
```

Figure 1: 程式初始化

1.1.2 Predict

基於動態方程估測下一時刻狀態, 為當前狀態加上控制向量與動態矩陣轉換下一狀態差生成下一刻狀態預測, 並計算下一時刻 P 誤差矩陣, 之後呼叫 Update 函數進行 filter , 演算法如 1 所示。

演算法參數對照程式碼：
$F = A$
$B = B$
$u = u$
$P = P$
$Q = Q$ 需要 tuning

Algorithm 1: Predict[1]

Input: state x , control u
Output: new state$\hat{x}$,P
1 $\hat{x} = Fx + Bu$
2 $P = FPF^T + Q$

轉換成程式碼：

```
25    def predict(self, u):
26        
27        # predict next state
28        self.x = self.A.dot(self.x) + self.B.dot(u)
29
30        # P Error matrix predict
31        self.P = self.A.dot(self.P).dot(self.A.T) + self.Q
32
```

Figure 2: Predict 函數

1.1.3 Update

經由前一時刻預測後的狀態與當前的狀態透過 Kalman gain(K) 調整 weight 重新計算出當前新的狀態, 並更新至 x 狀態及新的 P 協方矩陣, 演算法如 2 所示。
程式碼對照演算法：

1. 參數 $H = H$,R 需要 tuning
2. 計算 K 分兩階段計算 S 為 $(HPH^T + R)^{-1}$
3. 計算出 $K = PH^T S^{-1}$
4. $\hat{y}$ 對照程式 y 為計算量測狀態與預測狀態 margin, 程式碼為 $y = z - Hx$
5. 重新計算狀態 $x = x + Ky$, 由前步驟可得 K, y
6. $I - KH$ 和程式碼不同原因在於此方程式假設 K 值為最佳值, 但實際應用上並非總能計算出最優解, 所以應用上修改為 $P = (I - KH)P(I - KRK)^T + KRK^T$, 讓數值能更加穩定但缺點在於需要增加計算量, 經測試在本次作業兩者並無顯著差異 [1]。

Algorithm 2: Update[1]

Input: $\hat{x}$ P z
Output: $\hat{x}$ P
1 $\hat{y} = z - H\hat{x}$
2 $K = PH^T(HPH^T + R)^{-1}$
3 $\hat{x} = \hat{x} + K\hat{y}$
4 $P = (I - KH)P$
5 **return** $\hat{x}$, P

```python
def update(self, z):

    # Caluclate Kalman Gain
    S = self.H.dot(self.P).dot(self.H.T) +self.R
    self.K = self.P.dot(self.H.T).dot(np.linalg.inv(S))

    # Caluclate margin
    y = z - self.H.dot(self.x)

    # New state use Kalman gain
    self.x = self.x + self.K.dot(y)

    # New P Error matrix
    KH = self.K.dot(self.H)
    I_KH = self.I - KH
    self.P = (I_KH.dot(self.P.dot(I_KH.T)) + self.K.dot(self.R.dot(self.K.T)))
    return self.x, self.P
```

Figure 3: Update 函數

1.2 Q R tuning

Q 及 R 矩陣各別為協方差矩陣可定義為如下：

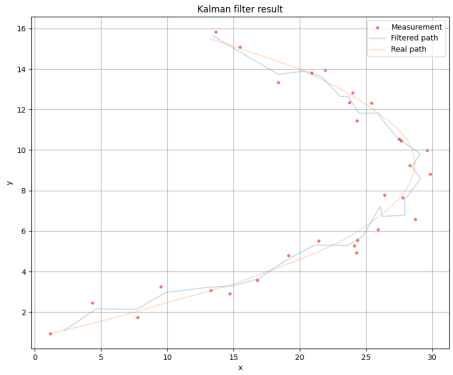
$$Q = \begin{bmatrix} {\sigma_x}^2 & cov(x,y) & cov(x,z) \\ cov(y,x) & {\sigma_y}^2 & cov(y,z) \\ cov(z,x) & cov(z,y) & {\sigma_z}^2 \end{bmatrix} \quad R = \begin{bmatrix} {\sigma_x}^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_y\sigma_x & {\sigma_y}^2 \end{bmatrix}$$

Control term : mean = 0 , var = 0.1

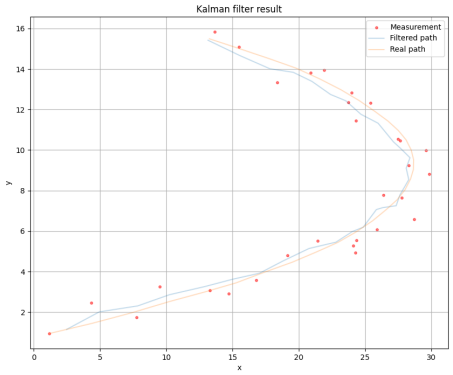
Measrement : mean = 0, var = 0.75

1.2.1 Analyze 1

將 Q R 矩陣皆初始設定為單位矩陣與已知輸入 variance, 則 Q = I*0.1 及 R = I*0.75 , 進行比較分析結果顯示以 Q = I*0.1 及 R = I*0.75 結果較佳，能有效抑制雜訊且由矩陣看出量測所造成的雜訊, 結果如圖下所示。



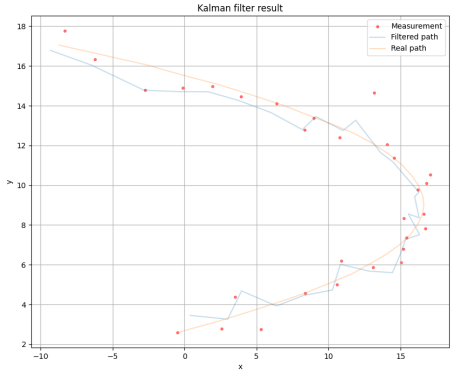
(a) Q R 為單位矩陣



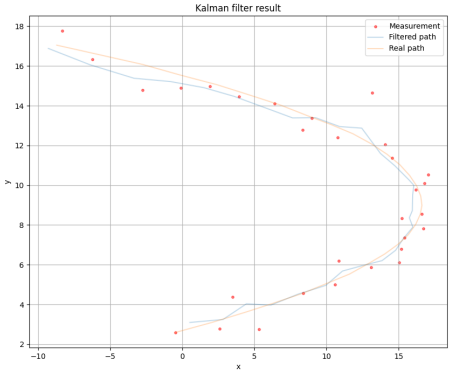
(b) Q R 以已知 var 做設計有效降低雜訊

1.2.2 Analyze 2

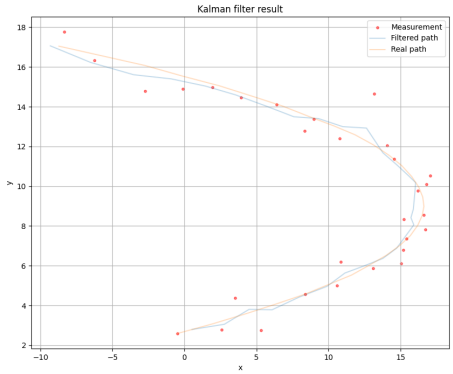
分析 2 以固定 $Q = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$, 調整 R $\begin{bmatrix} 0.75 & \rho * 0.75 \\ \rho * 0.75 & 0.75 \end{bmatrix}$ 矩陣中的 ρ 分析量測變異數 x 和 y 之間相關性, 設定條件 $-1 < \rho < 1$ 之間依序調整驗證相關性, 經驗證結果 $\rho = 0$ 最接近 ground truth 所以在量測雜訊中 x y 為非相關, 結果如圖下所示。



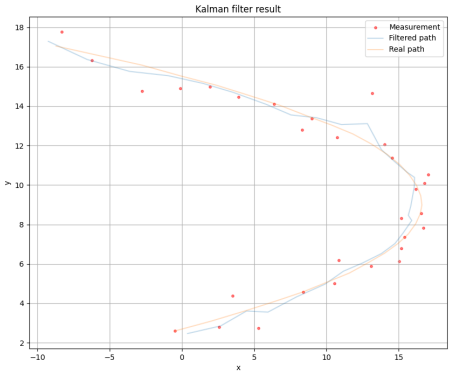
(c) $\rho = 0.99$



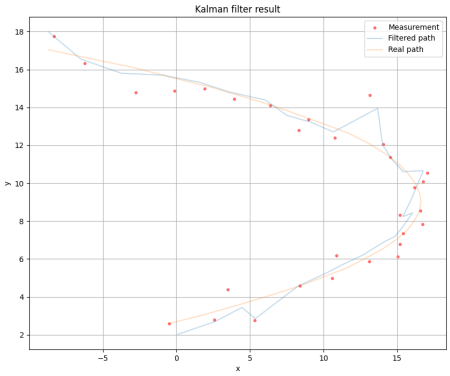
(d) $\rho = 0.5$



(e) $\rho = 0$



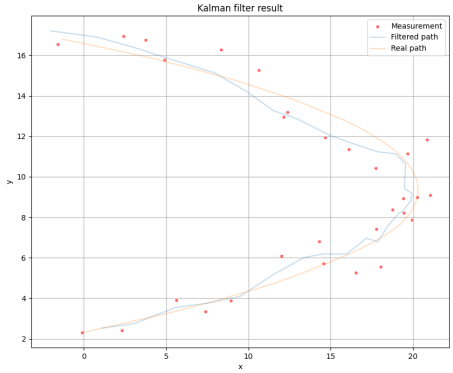
(f) $\rho = -0.5$



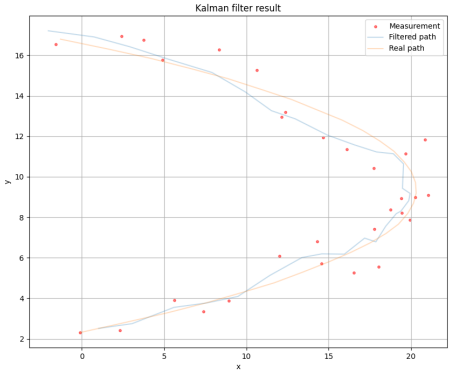
(g) $\rho = -0.99$

1.2.3 Analyze 3

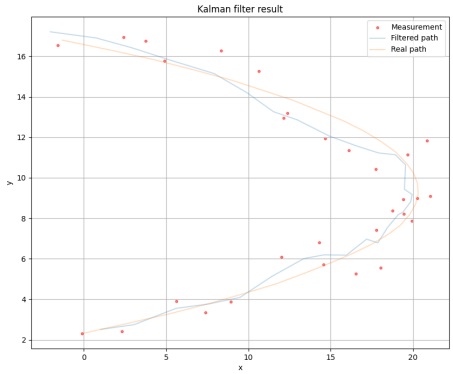
分析 3 固定 $R = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$, 調整 $Q = \begin{bmatrix} 0.1 & 0 & \rho * 0.1 \\ 0 & 0.1 & \rho * 0.1 \\ \rho * 0.1 & \rho * 0.1 & 0.1 \end{bmatrix}$, 設定條件 $-1 < \rho < 1$ 之間依序調整驗證 x y 和 yaw 之間的相關性結果為非相關, 因此結果無明顯誤差變化結果如圖所示。



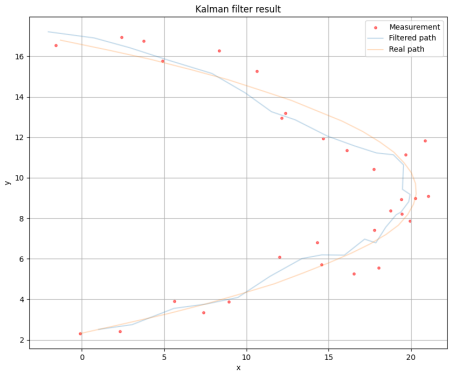
(h) $\rho = 0.99$



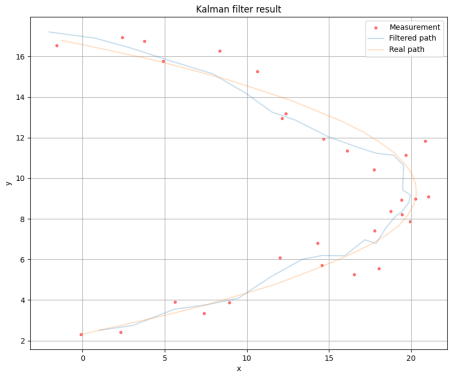
(i) $\rho = 0.5$



(j) $\rho = 0$



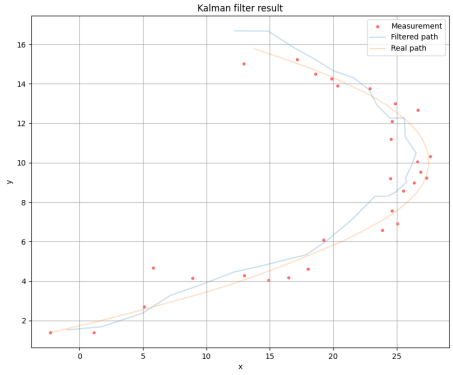
(k) $\rho = -0.5$



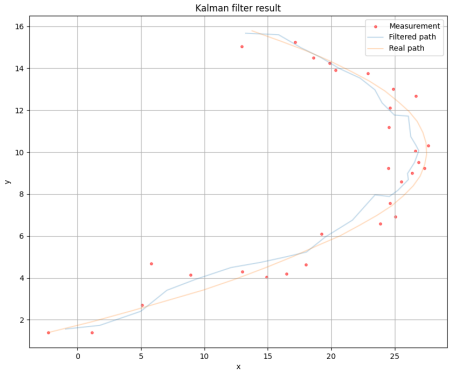
(l) $\rho = 0.99$

1.3 Analyze 4

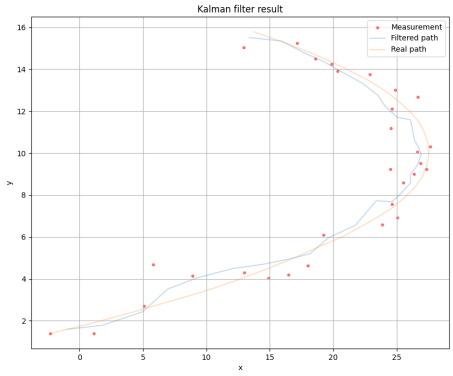
分析 4 固定 $R = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$, 調整 $Q = \begin{bmatrix} 0.1 & \rho * 0.10 & \\ \rho * 0.1 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$, 設定條件 $-1 < \rho < 1$ 之間依序驗證預測 x y 之間的相關性, 結果顯示 $\rho = 0.5$ 濾波效果較佳, 結果如圖下所示。



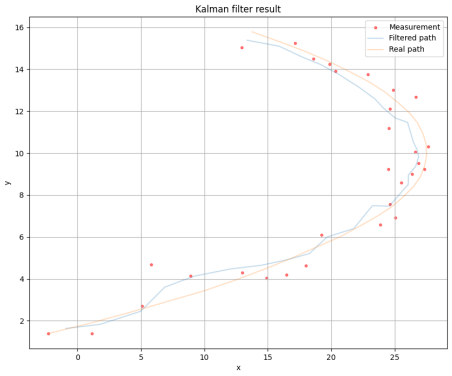
(m) $\rho = 0.99$



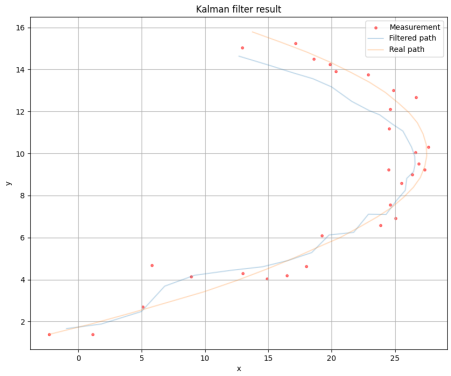
(n) $\rho = 0.5$



(o) $\rho = 0$



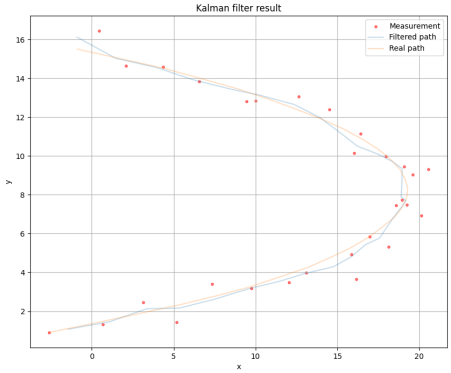
(p) $\rho = -0.5$



(q) $\rho = 0.99$

1.3.1 Analyze Q R result

經分結果所設計的預測協方矩陣為 $Q = \begin{bmatrix} 0.1 & 0.05 & 0 \\ 0.05 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$, 則 $R = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.75 \end{bmatrix}$, 經分析量測的 unsertainty 較大所以量測的 weight 較小, 所以較依賴預測的狀態, 結果如圖下示。



(r) Q R 設計結果

References

[1] Labbe, R. (2014). Kalman and bayesian filters in python. Chap, 7(246), 4.

[2] Faragher, R. (2012). Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. IEEE Signal processing magazine, 29(5), 128-132.