Assignment I: Advanced Robot Navigation (RBE 595)

Pradnya Sushil Shinde

March 8, 2024

1 Kalman Filter

1.1 TASK I: Kalman Filter Equations

The continuous time model of a dynamic system can be defined as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(\mathbf{t}) + \mathbf{B}\mathbf{u}(\mathbf{t})$$

where $\mathbf{x(t)}$ is the system state space defined as $x = [p\dot{p}]$ where p is position.

We have access to the position and velocity data of the drone, therefore we will assume a constant acceleration model. The input $\mathbf{u(t)}$, in this case, is the net force.

$$u = F_{net} = ma$$

$$a = u/m$$

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = A \begin{bmatrix} \dot{X}_t \\ \dot{Y}_t \\ \dot{Z}_t \\ \ddot{X}_t \\ \ddot{Y}_t \\ \ddot{Z}_t \end{bmatrix} + B \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

We can define A and B as:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix}$$

To derive the discrete-time model given by,

$$\mathbf{x_t} = \mathbf{F x_{t-1}} + \mathbf{G u_t} + \mathbf{Q}$$

where **F** is State Transition Matrix, **G** is the Input Transition Matrix and **Q** is Process Noise Matrix.

$$F = e^{A\Delta t} = I + A\Delta t + \frac{(A\Delta t)^2}{2!} + \frac{(A\Delta t)^3}{3!} + ...$$

A^2 is 0, which gives us:

$$F = I + A\Delta t$$

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$G = \int_0^{\Delta t} e^{At} dt B = (\Delta t I + A(\frac{\Delta^2}{2}))B$$

$$G = \begin{bmatrix} \Delta t & 0 & 0 & \Delta t^2/2 & 0 & 0 \\ 0 & \Delta t & 0 & 0 & \Delta t^2/2 & 0 \\ 0 & 0 & \Delta t & 0 & 0 & \Delta t^2/2 \\ 0 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & \Delta t \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} \Delta t^2/2m & 0 & 0 \\ 0 & \Delta t^2/2m & 0 \\ 0 & 0 & \Delta t^2/2m \\ \Delta t/m & 0 & 0 \\ 0 & \Delta t/m & 0 \\ 0 & 0 & \Delta t/m \end{bmatrix}$$

$\Delta t^2/m$ is the factor which when multiplied by the input force u causes change in the position x. This can be derived through:

$$F = u = ma$$

$$u = m\frac{x}{t^2}$$

$$x = u\frac{\mathbf{t^2}}{\mathbf{m}}$$

$\Delta t/m$ is the factor which when multiplied by the input force u causes change in the velocity v. This can be derived through:

$$F = u = ma$$
$$u = m\frac{v}{t}$$
$$v = u\frac{\mathbf{t}}{\mathbf{m}}$$

where t will be Δt during state transition.

The Estimate Uncertainty matrix represented by $\mathbf{P}$ for the above state space model is given below:

$$\mathbf{P} = \begin{bmatrix} P_{xx} & 0 & 0 & 0 & 0 & 0 \\ 0 & P_{yy} & 0 & 0 & 0 & 0 \\ 0 & 0 & P_{zz} & 0 & 0 & 0 \\ 0 & 0 & 0 & P_{v_x v_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & P_{v_y v_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & P_{v_z v_z} \end{bmatrix}$$

The measurement matrix $\mathbf{H}$ will depend on the type of measurement taken. For the above State Space Model, if the measurement parameter is position then:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

If the measurement parameter is velocity then:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In summary, the Kalman Filter is essentially modeled using the following set of equations:

1. PREDICT STEP
   STATE EXTRAPOLATION
   $$\mathbf{\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n} \tag{1}$$
   UNCERTAINITY EXTRAPOLATION
   $$\mathbf{P_{n+1,n} = FP_{n,n}F^T + Q} \tag{2}$$
2. UPDATE STEP
   KALMAN GAIN
   $$\mathbf{K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1}} \tag{3}$$

UPDATE MEASUREMENT

$$\mathbf{\hat{x}_{n,n}} = \mathbf{\hat{x}_{n,n-1}} + \mathbf{K_n}(\mathbf{z_n} - \mathbf{H}\mathbf{\hat{x}_{n,n-1}}) \tag{4}$$

UPDATE UNCERTAINITY

$$\mathbf{P_{n,n}} = (\mathbf{I} - \mathbf{K_n}\mathbf{H})\mathbf{P_{n,n-1}}(\mathbf{I} - \mathbf{K_n}\mathbf{H})^{\mathbf{T}} + \mathbf{K_n}\mathbf{R_n}\mathbf{K_n^T} \tag{5}$$

$\mathbf{x}$ - State Vector
$\mathbf{z_n}$ - measurement at time n
$\mathbf{u_n}$ - control input at time n
K - Kalman Gain
F - State Transition Matrix
G - Input Control Matrix
H - Measurement matrix
P - Estimate Uncertainty matrix
Q - Process Noise Covariance matrix
R - Measurement Noise Covariance matrix

1.2 TASK II

Import the necessary library.

```python
# Kalman Filter Implementation
import numpy as np
import matplotlib.pyplot as plt
import argparse
from mpl_toolkits.mplot3d import Axes3D
```

Define a class "Kalman Filter" that implements the state estimation methods.

```python
# Define Kalman Filter class
class KalmanFilter:
    def __init__(self):
        self.m = 0.027 # Mass of the Drone

    def computeF_G(self, A, B, dt):
        # To derive discrete time model from the continuous-time model
        # State Space: xdot = Ax + Bu
        # where x = [p pdot] and p is position

        # F = I + A*dt
        F = np.eye(6) + A*dt

        # G = (dt*I + A*((dt*dt)/2))*B
        G = (np.eye(6)*dt + (A*(dt**2))) @ B

        return F, G

```

```python
    def predict(self,F, G, xhat_n, u_n, P_n, Q):
        # Prediction step
        xhat_pred = F @ xhat_n + G @ u_n
        p_pred = F @ P_n @ F.T + Q
        return xhat_pred, p_pred

    def computeKGain(self, xhat_pred, p_pred, z_n, H, R):
        # Compute Kalman Gain and Displacement
        K = p_pred @ H.T @ np.linalg.inv(H @ p_pred @ H.T + R)
        y = z_n - H @ xhat_pred
        return y, K
```

We have been given data related to Position and Velocity measurements ($\mathbf{z}$) taken at different time stamps ($\mathbf{t}$) and each having a net force as control input ($\mathbf{u}$). We will read the data files and exract relevant values.

```python
# Read data from CSV file
def read_data(data_file):
    data = np.loadtxt(data_file, delimiter=',')
    t = data[:,0]
    u = data[:,1:4]  # Input (force)
    z = data[:,4:7]  # Measurement (position or velocity)
    return t, u, z
```

Defining a Kalman Filter process flow that combines the Measurement and Update steps.

```python
# Run Kalman filter on data
def run_kalman_filter(t, u, z, R, Q, StateParam):
    # Initialise a KalmanFilter class object
    kf = KalmanFilter()
    n = len(t)
    A = np.array([[0, 0, 0, 1, 0, 0],
                  [0, 0, 0, 0, 1, 0],
                  [0, 0, 0, 0, 0, 1],
                  [0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0]])
    B = np.array([[0, 0, 0],
                  [0, 0, 0],
                  [0, 0, 0],
                  [1/kf.m, 0, 0],
                  [0, 1/kf.m, 0],
                  [0, 0, 1/kf.m]])
    P = np.zeros((n, 6, 6))
    P[0] = np.diag([0.01, 0.01, 0.01, 0.05, 0.05, 0.05])
    xhat = np.zeros((n, 6))

    if StateParam == 'Position':
        xhat[0] = np.concatenate([z[0], np.zeros(3)])  # Initial State: [x,
 y, z, 0, 0, 0]
```

```
        H = np.array([[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]])
    elif StateParam == 'Velocity':
        xhat[0] = np.concatenate([np.zeros(3),z[0]]) # Initial State: [0, 0, 0, xdot, ydot, zdot]
        H = np.array([[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]])

    for i in range(1,n):
        dt = t[i] - t[i-1]
        F, G = kf.computeF_G(A, B, dt)

        # Prediction Step
        xhat_n = xhat[i-1]
        u_n = u[i-1]
        P_n = P[i-1]
        xhat_pred, p_pred = kf.predict(F, G, xhat_n, u_n, P_n, Q)

        z_i = z[i]
        y, K = kf.computeKGain(xhat_pred, p_pred, z_i, H, R)

        # Update Step
        xhat[i] = xhat_pred + K @ y  # Update State
        P[i] = (np.eye(6) - K @ H) @ p_pred # Update Estimate

    return xhat, P
```

Visualize the data as a 3D model.

```
def visualize(xhat):
    p_hat = xhat[:, :3]
    fig = plt.figure(figsize=(10,6))
    ax = plt.axes(projection='3d')
    ax.scatter3D(p_hat[:, 0], p_hat[:, 1], p_hat[:, 2], s=3)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    plt.show()
```

Motion Tracking process flow is encapsulated within the "main()" fucntion as below:

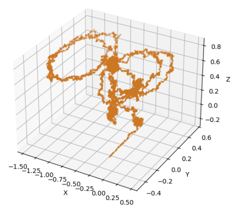
```
# Main function
def main():
    Parser = argparse.ArgumentParser()
    Parser.add_argument("--StateParam", default='Position', help="Provide the State Parameter for which Kalman Filter should be implemented.")
    Parser.add_argument("--FileType", default="mocap", help="Choose from: mocap, low_noise, high_noise, velocity")
    Parser.add_argument("--Visualize", default="Yes", help="Bool Value for Visualizing Results(Yes/No)")

```

```
    Args = Parser.parse_args()
    StateParam = Args.StateParam
    FileType = Args.FileType
    Viz = Args.Visualize
    # Read data
    base_file = "kalman_filter_data_"
    data_file = base_file + FileType + '.txt'
    t, u, z = read_data(data_file)

    # Q: Process Noise Variance
    # R: Measurement Noise Variance
    # LOW Q: Low Variation in State Transition, HIGH Q: High Variation in State Transition
    # LOW R: Expects Less Noise, HIGH R: Expects High Noise
    if FileType == 'mocap':
        Q = np.eye(6)*((0.003)**2)
        R = np.eye(3)*((0.01)**2)
    elif FileType == 'low_noise':
        Q = np.eye(6)*((0.01)**2)
        R = np.eye(3)*((0.5)**2)
    elif FileType == 'high_noise':
        Q = np.eye(6)*((0.005)**2)
        R = np.eye(3)*((1.5)**2)
    elif FileType == 'velocity':
        Q = np.eye(6)*((0.01)**2)
        R = np.eye(3)*((0.1)**2)

    xhat, P = run_kalman_filter(t, u, z, R, Q, StateParam)

    if Viz == "Yes":
        visualize(xhat)

if __name__ == "__main__":
    main()
```

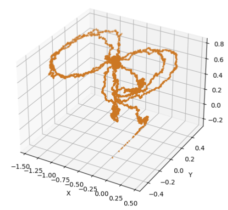
2 Analysis

Choosing optimal values of $\mathbf{Q}$ and $\mathbf{R}$ is crucial. Suppose we have a system with high noise measurements, if the process noise covariance $\mathbf{Q}$ is set too low, the filter may overly rely on noisy measurements, leading to a less accurate estimation of the true state. Conversely, setting it too high may result in overly conservative predictions, smoothing out the estimates excessively and potentially masking the true dynamics of the system. Figure 1 shows the visualization of how Q and R might affect high-noise measurement data.

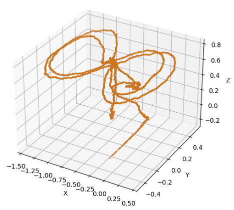
Given the analysis above, Figure 2 shows the visualization of the Kalman Filter model with optimal values of Q and R in each case.



(a) High Q (0.05) & High R (1.5)

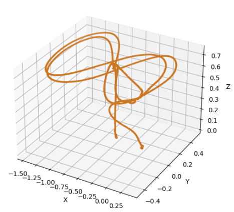


(b) Low Q (0.001) & Low R (0.1)

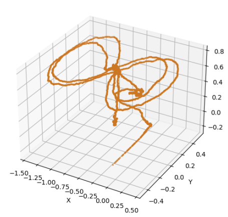


(c) Optimal Q (0.005) & R(1)

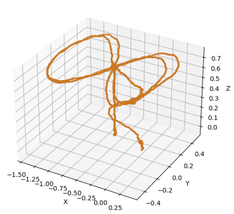
Figure 1: **Variations in Noise Covariance**



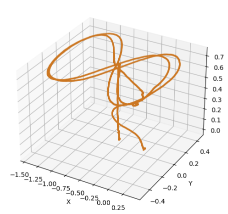
(a) Motion Capture



(b) High Noise



(c) Low Noise



(d) Velocity

Figure 2: **Kalman Filter Responses**