MODULE 5: NETWORKS AND GRAPHICAL MODELS

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 2.2 –KALMAN FILTERING: TRACKING LOCATION OF AN OBJECT IN 3D

MIT xPRO

CASE STUDY ACTIVITY TUTORIAL

CASE STUDY 2.2 –KALMAN FILTERING: TRACKING LOCATION OF AN OBJECT IN 3D

Faculty: Guy Bresler

In this document, we walk through some helpful tips to get you started with tracking the state of an object in 3 dimensions, using Kalman Filtering, with gravity acting on the object. In this tutorial, we provide examples and some pseudo-code for the following programming environment: **Python**. We cover the following:

Topics

The Model

Please refer to the attached document ("Kalman-Model-3D-Tracking") for a detailed description of the model we will use for this case study. The rest of this document assumes familiarity with the model details.

Generating Data

We need to synthetically generate noisy measurements of the location of the object (ball) in 3 dimensions. In Python, this can be done in the following manner.

```
# Time step
dt = 0.01

# total number of measurements
m = 200
```

```
# positions at start
px= 0.0
py= 0.0
pz= 1.0

# velocities at start
vx = 5.0
vy = 3.0
vz = 0.0

# Drag Resistance Coefficient
c = 0.1

# Damping
d = 0.9

# Arrays to store location measurements
Xr=[]
Yr=[]
Zr=[]

# generating data
for i in range(0, m):

    # update acceleration (deceleration), velocity, position in x direction
    accx = -c*vx**2
    vx += accx*dt
    px += vx*dt

    # update acceleration (deceleration), velocity, position in y direction
    accy = -c*vy**2
    vy += accy*dt
    py += vy*dt

    # update acceleration, velocity, position in x direction
    accz = -9.806 + c*vz**2
    vz += accz*dt
    pz += vz*dt

    # if the object is about to hit the base,
    # change direction, with damping
    if pz<0.01:
        vz=-vz*d
        pz+=0.02

    # add to the arrays storing locations
    Xr.append(px)
    Yr.append(py)
    Zr.append(pz)
```

```
# Add random noise to measurements
# Standard Deviation for noise
sp= 0.1

Xm = Xr + sp * (np.random.randn(m))
Ym = Yr + sp * (np.random.randn(m))
Zm = Zr + sp * (np.random.randn(m))

# stack the measurements together for ease of later use
measurements = np.vstack((Xm,Ym,Zm))
```

Visualizing Data

In order to visualize the data generated, we will need the **matplotlib** library in Python. Once installed, it can be used to visualize the data in the following manner:

```
fig = plt.figure()
3dplot = fig.add_subplot(111, projection='3d')
3dplot.scatter(Xm, Ym, Zm, c='red')
3dplot.set_xlabel('$x$')
3dplot.set_ylabel('$y$')
3dplot.set_zlabel('$z$')
plt.title('Noisy 3D Ball-Location observations')
plt.show()
```

Initializing Variables

We can initialize the other variables and matrices discussed in the model in the following manner. Note that we will need the **numpy** library for these.

```
import numpy as np

# Identity matrix
I = np.eye(9)

# state matrix
x = np.matrix([0.0, 0.0, 1.0, 5.0, 3.0, 0.0, 0.0, 0.0, -9.81]).T

# P matrix
P = 100.0*np.eye(9)

# A matrix
A = np.matrix([[1.0, 0.0, 0.0, dt, 0.0, 0.0, 1/2.0*dt**2, 0.0, 0.0],
               [0.0, 1.0, 0.0, 0.0,  dt, 0.0, 0.0, 1/2.0*dt**2, 0.0],
               [0.0, 0.0, 1.0, 0.0, 0.0,  dt, 0.0, 0.0, 1/2.0*dt**2],
               [0.0, 0.0, 0.0, 1.0, 0.0, 0.0,  dt, 0.0, 0.0],
               [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,  dt, 0.0],
               [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,  dt],
```

```
                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0],
                    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]])

# H matrix
H = np.matrix([[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                    [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]])

# R matrix
r = 1.0

R = np.matrix([[r, 0.0, 0.0],
                    [0.0, r, 0.0],
                    [0.0, 0.0, r]])

# Q, G matrices
s = 8.8

G = np.matrix([[1/2.0*dt**2],
                    [1/2.0*dt**2],
                    [1/2.0*dt**2],
                    [dt],
                    [dt],
                    [dt],
                    [1.0],
                    [1.0],
                    [1.0]])
Q = G*G.T*s**2
```

Kalman Filtering Algorithm

We can now run the Kalman Filtering algorithm with the following lines of code in Python. We will continue to need **numpy.**

```
# The Following variables will store the results, at each iteration:
xt = []
yt = []
zt = []
dxt= []
dyt= []
dzt= []
ddxt=[]
ddyt=[]
ddzt=[]
Zx = []
Zy = []
```

```
        Zz = []
        Px = []
        Py = []
        Pz = []
        Pdx= []
        Pdy= []
        Pdz= []
        Pddx=[]
        Pddy=[]
        Pddz=[]
        Kx = []
        Ky = []
        Kz = []
        Kdx= []
        Kdy= []
        Kdz= []
        Kddx=[]
        Kddy=[]
        Kddz=[]
################################################

onFloor = False
for i in range(0, m):

    # Model the direction switch, when hitting the plate
    if x[2]<0.02 and not onFloor:
        x[5] = -x[5]
        onFloor=True

    # Prediction
    # state prediction
    x = A*x + B*u

    # Project the error covariance ahead
    P = A*P*A.T + Q

    # Update
    # Kalman Gain
    S = H*P*H.T + R
    K = (P*H.T) * np.linalg.pinv(S)

    # Update the estimate via z
    Z = measurements[:,i].reshape(H.shape[0],1)
    y = Z - (H*x)
    x = x + (K*y)

    # error covariance
    P = (I - (K*H))*P
```

```
# Storing results
xt.append(float(x[0]))
yt.append(float(x[1]))
zt.append(float(x[2]))
dxt.append(float(x[3]))
dyt.append(float(x[4]))
dzt.append(float(x[5]))
ddxt.append(float(x[6]))
ddyt.append(float(x[7]))
ddzt.append(float(x[8]))

Zx.append(float(Z[0]))
Zy.append(float(Z[1]))
Zz.append(float(Z[2]))
Px.append(float(P[0,0]))
Py.append(float(P[1,1]))
Pz.append(float(P[2,2]))
Pdx.append(float(P[3,3]))
Pdy.append(float(P[4,4]))
Pdz.append(float(P[5,5]))
Pddx.append(float(P[6,6]))
Pddy.append(float(P[7,7]))
Pddz.append(float(P[8,8]))
Kx.append(float(K[0,0]))
Ky.append(float(K[1,0]))
Kz.append(float(K[2,0]))
Kdx.append(float(K[3,0]))
Kdy.append(float(K[4,0]))
Kdz.append(float(K[5,0]))
Kddx.append(float(K[6,0]))
Kddy.append(float(K[7,0]))
Kddz.append(float(K[8,0]))
```

Visualizing Results

We can now visualize the results using the following lines of code in Python. Note that we will need the **matplotlib** library for this.

```
#####################################################

    # Plots
    #State Estimates
    plt.subplot(311)
    plt.title('Location, Velocity, Acceleration Estimates')
    plt.plot(range(len(measurements[0])),xt, label='$x$')
    plt.plot(range(len(measurements[0])),yt, label='$y$')
```

```
plt.plot(range(len(measurements[0])),zt, label='$z$')
plt.legend(loc='right' )

plt.subplot(312)
plt.plot(range(len(measurements[0])),dxt, label='$v_x$')
plt.plot(range(len(measurements[0])),dyt, label='$v_y$')
plt.plot(range(len(measurements[0])),dzt, label='$v_z$')
plt.legend(loc='right')

plt.subplot(313)
plt.plot(range(len(measurements[0])),ddxt, label='$ax$')
plt.plot(range(len(measurements[0])),ddyt, label='$ay$')
plt.plot(range(len(measurements[0])),ddzt, label='$az$')
plt.legend(loc='right')

plt.xlabel('Step')

# Location in 2D (z, y)
plt.subplot(311)
plt.plot(xt,yt, label='Estimate')
plt.scatter(Xm,Ym, label='Measurement', c='red', s=30)
plt.plot(Xr, Yr, label='Real')
plt.title('Location Tracking on the 2D-Planes')
plt.legend(loc='best')
plt.xlabel('$x$')
plt.ylabel('$y$')

plt.subplot(312)
plt.plot(xt,zt, label='Estimate')
plt.scatter(Xm,Zm, label='Measurement', c='red', s=30)
plt.plot(Xr, Zr, label='Real')
plt.legend(loc='best')
plt.xlabel('$x$')
plt.ylabel('$z$')

plt.subplot(313)
plt.plot(yt,zt, label='Estimate')
plt.scatter(Ym,Zm, label='Measurement', c='red', s=30)
plt.plot(Yr, Zr, label='Real')
plt.legend(loc='best')
plt.axhline(0, color='k')
plt.xlabel('$y$')
plt.ylabel('$z$')

# Location in 3D (X, Y, Z)
ax = fig.add_subplot(111, projection='3d')
3dplt.plot(xt,yt,zt, label='Estimate')
3dplt.scatter(Xm,Ym,Zm, label='Measurement', c='red')
```

```
3dplt.plot(Xr, Yr, Zr, label='Real')
3dplt.set_xlabel('$x$')
3dplt.set_ylabel('$y$')
3dplt.set_zlabel('$z$')
3dplt.legend()
plt.title('3D Location Tracking')

plt.show()
```

References

https://balzer82.github.io/Kalman