Tony Smoragiewicz

RSS Ex3

In [28]:
```
import numpy as np
from numpy.linalg import multi_dot
from numpy.linalg import inv
```

1-D displacement

In [29]:
```
def kalman_1d(belief, sigma_v, sigma_w, u, z):
    x = belief[0]
    var = belief[1]

    # prediction
    x_pred = x + u
    var_pred = var + sigma_v

    # innovation
    nu = z - x_pred
    k = var_pred/(var_pred+sigma_w)

    # update
    x_new = x_pred + k*nu
    var_new = (1 - k) * var_pred

    posterior = [x_new, var_new]

    return posterior
```

In [30]:
```
sigma_v = 0.04
sigma_w = 0.01
U = [-0.5, 1.2, 0.3]
Z = [-0.7, 0.6, 0.95]
belief = [0, 1]

for i in range(3):
    belief = kalman_1d(belief, sigma_v, sigma_w, U[i], Z[i])
print(np.around(belief, 5))
```

```
[0.93862 0.00829]
```

2-D displacement

In [31]:
```
def kalman_nd(belief, Sigma_v, Sigma_w, u, z):
    [r, c] = np.shape(belief)
    x = np.array(belief[:, 0])
    var = belief[:, 1:]

    # prediction
    x_pred = x + u
    var_pred = var + Sigma_v

    # innovation
    nu = z - x_pred
    K = var_pred*np.linalg.inv(var_pred+Sigma_w)

    # update
    x_new = x_pred + np.matmul(K, nu)
    var_new = np.matmul((np.identity(r) - K), var_pred)

    posterior = np.vstack((x_new, var_new))

    return posterior.T
```

In [32]:
```
sigma_v = np.array([[0.04, 0],
                    [0, 0.09]])

sigma_w = np.array([[0.01, 0],
                    [0, 0.02]])

U = np.array([[-0.5, 1.2, 0.3],
              [0.3, -0.6, 0.3]])

Z = np.array([[-0.7, 0.6, 0.95],
              [0.3, 0.0, 0.15]])

belief = np.array([[0, 1, 0],
                   [0, 0, 1]])

for i in range(3):
    belief = kalman_nd(belief, sigma_v, sigma_w, U[:, i], Z[:, i])
print(np.around(belief, 5))
```

```
[[0.93862 0.00829 0.     ]
 [0.16634 0.      0.01685]]
```

1-D Motion

- Prediction Step

$$X^+_{t+1} = FX_t + Gu_t$$

$$\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}^+_{t+1} = \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}_t + \frac{1}{m} \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \\ 1 \end{bmatrix} u_t$$

$$P^+_{t+1} = FP_tF^T + V$$

$$\begin{bmatrix} \sigma_{pos} & 0 & 0 \\ 0 & \sigma_{vel} & 0 \\ 0 & 0 & \sigma_{acc} \end{bmatrix}^+_{t+1} = \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \sigma_{pos} & 0 & 0 \\ 0 & \sigma_{vel} & 0 \\ 0 & 0 & \sigma_{acc} \end{bmatrix}_t \begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}^T + Noise$$

- Innovation Step

$$\nu = z_{t+1} - HX^+_{t+1}$$

$$\nu = z_{t+1} - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}^+_{t+1}$$

$$K = P^+_{t+1}H^T(HP^+_{t+1}H^T + W)^{-1}$$

$$K = \begin{bmatrix} \sigma_{pos} & 0 & 0 \\ 0 & \sigma_{vel} & 0 \\ 0 & 0 & \sigma_{acc} \end{bmatrix}^+_{t+1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \sigma_{pos} & 0 & 0 \\ 0 & \sigma_{vel} & 0 \\ 0 & 0 & \sigma_{acc} \end{bmatrix}^+_{t+1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \sigma_w^2 \right)^{-1}$$

- Update Step

$$X_{t+1} = X^+_{t+1} + K\nu$$

$$P_{t+1} = (I - KH)P^+_{t+1}$$

$$P_{t+1} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - K \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} \sigma_{pos} & 0 & 0 \\ 0 & \sigma_{vel} & 0 \\ 0 & 0 & \sigma_{acc} \end{bmatrix}^+_{t+1}$$

In [33]:
```
def kalman_1dmotion(belief, Sigma_v, Sigma_w, u, z, F, G, H):
    [r, c] = np.shape(belief)
    x = np.array(belief[:, 0])
    x = np.reshape(x, [r, 1])
    var = belief[:, 1:]

    # prediction
    Gu = np.dot(G, u)
    x_pred = np.matmul(F, x) + Gu
    var_pred = multi_dot([F, var, F.T]) + Sigma_v

    # innovation
    nu = z - np.matmul(H, x_pred)
    HPHpW = multi_dot([H, var_pred, H.T]) + Sigma_w
    K = multi_dot([var_pred, H.T, inv(HPHpW)])

    # update
    x_new = x_pred + K*nu
    I = np.identity(r)
    KH = np.matmul(K, H)
    var_new = np.matmul((I - KH), var_pred)

    posterior = np.hstack((x_new, var_new))

    return posterior
```

In [34]:
```
sigma_v = np.array([[0.02, 0, 0],
                    [0, 0.03, 0],
                    [0, 0, 0]])

sigma_w = 0.01

U = np.array([-0.2, 0.0, 0.1])

Z = np.array([0.4, 0.9, 0.8])

belief = np.array([[0, 1, 0, 0],
                   [0.5, 0, 2, 0],
                   [0, 0, 0, 0]])

dt = 1

F = np.array([[1, dt, 0],
             [0, 1, 0],
             [0, 0, 0]])

m = 1
G = 1/m * np.array([[0.5*dt**2], [dt], [1]])

H = np.array([1, 0, 0])
H = np.reshape(H, [1, 3])

for i in range(3):
    belief = kalman_1dmotion(belief, sigma_v, sigma_w, U[i], Z[i], F, G, H)
print(np.around(belief, 5))
```

```
[[0.8505  0.0092  0.00608 0.     ]
 [0.20138 0.00608 0.05058 0.     ]
 [0.1     0.      0.      0.     ]]
```

2-D displacement with nonlinear measurements

- Prediction Step

$$X_{t+1}^{+} = X_t + u_t$$

$$\begin{bmatrix} x \\ y \end{bmatrix}_{t+1}^{+} = \begin{bmatrix} x \\ y \end{bmatrix}_{t} + u_t$$

$$P_{t+1}^{+} = P_t + \Sigma_v$$

$$\begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t} + \begin{bmatrix} 0.04 & 0 \\ 0 & 0.09 \end{bmatrix}$$

- Innovation Step

$$\nu = z_{t+1} - h(X_{t+1}^{+})$$

$$\nu = z_{t+1} - \sqrt{{x_{t+1}^{+}}^2 + {y_{t+1}^{+}}^2}$$

$$K = P_{t+1}^{+} H_x^T (H_x P_{t+1}^{+} H_x^T + H_w W H_w^T)^{-1}$$

$$h = x^2 + y^2 + \sigma_w^2 \quad H_x = \frac{\partial h}{\partial x} = \begin{bmatrix} 2x & 2y \end{bmatrix} \quad H_w = \frac{\partial h}{\partial w} = \begin{bmatrix} 1 \end{bmatrix}$$

$$K = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} \begin{bmatrix} 2x \\ 2y \end{bmatrix} \left(\begin{bmatrix} 2x & 2y \end{bmatrix} \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} \begin{bmatrix} 2x \\ 2y \end{bmatrix} + \sigma_w^2 \right)^{-1}$$

- Update Step

$$X_{t+1} = X_{t+1}^{+} + K\nu$$

$$P_{t+1} = P_{t+1}^{+} - K H_x P_{t+1}^{+}$$

$$P_{t+1} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} - K \begin{bmatrix} 2x & 2y \end{bmatrix} \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+}$$

In [35]:
```
def kalman_nonlinear(belief, Sigma_v, Sigma_w, u, z):
    [r, c] = np.shape(belief)
    x = np.array(belief[:, 0])
    var = belief[:, 1:]

    # prediction
    x_pred = x + u
    var_pred = var + Sigma_v

    # innovation
    h = x_pred[0]**2 + x_pred[1]**2
    nu = z - h
    Hx = np.array([2*x_pred[0], 2*x_pred[1]])
    HxT = np.reshape(Hx, (2, 1))
    HPH = multi_dot([Hx, var_pred, HxT])
    PH = np.matmul(var_pred, HxT)
    K = PH/(HPH+Sigma_w)

    # update
    x_new = x_pred + K.T*nu
    var_new = var_pred - multi_dot([K*Hx, var_pred])
    posterior = np.hstack((x_new.T, var_new))

    return posterior
```

In [36]:
```
sigma_v = np.array([[0.04, 0],
                    [0, 0.09]])

sigma_w = 0.01

U = np.array([[-0.5, 1.2, 0.3],
              [0.3, -0.6, 0.3]])

Z = np.array([0.6, 0.4, 1.0])

belief = np.array([[0, 1, 0],
                   [0, 0, 1]])

for i in range(3):
    belief = kalman_nonlinear(belief, sigma_v, sigma_w, U[:, i], Z[i])
print(belief)
```

```
[[ 0.96233304  0.01089887 -0.03425474]
 [ 0.27211962 -0.03425474  0.14176062]]
```

Light-dark domain

- Prediction Step

$$X_{t+1}^{+} = X_t + u_t$$

$$\begin{bmatrix} x \\ y \end{bmatrix}_{t+1}^{+} = \begin{bmatrix} x \\ y \end{bmatrix}_{t} + u_t$$

$$P_{t+1}^{+} = P_t + \Sigma_v$$

$$\begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t} + \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

- Innovation Step

$$\nu = z_{t+1} - X_{t+1}^{+}$$

$$\nu = \begin{bmatrix} z_x \\ z_y \end{bmatrix}_{t+1} - \begin{bmatrix} x \\ y \end{bmatrix}_{t+1}^{+}$$

$$K = P_{t+1}^{+}(P_{t+1}^{+} + \Sigma_w)^{-1}$$

$$K = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} \left(\begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+} + \begin{bmatrix} \sigma_w^2(x) & 0 \\ 0 & \sigma_w^2(x) \end{bmatrix} \right)^{-1} \quad \sigma_w^2(x) = \frac{1}{2}(5-x)^2 + 0.01$$

- Update Step

$$X_{t+1} = X_{t+1}^{+} + K\nu$$

$$P_{t+1} = (I - K)P_{t+1}^{+}$$

$$P_{t+1} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - K \right) \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix}_{t+1}^{+}$$

In [37]:
```
def kalman_ld(belief, Sigma_v, Sigma_w, u, z):
    [r, c] = np.shape(belief)
    x = np.array(belief[:, 0])
    var = belief[:, 1:]

    # prediction
    x_pred = x + u
    var_pred = var + Sigma_v

    # innovation
    nu = z - x_pred
    K = var_pred*np.linalg.inv(var_pred+Sigma_w)

    # update
    x_new = x_pred + np.matmul(K, nu)
    var_new = np.matmul((np.identity(r) - K), var_pred)

    posterior = np.vstack((x_new, var_new))

    return posterior.T
```

In [38]:
```
sigma_v = np.array([[0.01, 0],
                    [0, 0.01]])

U = np.array([[1.0, 2.0, -5.0],
              [-1.0, -1.0, 0.0]])

Z = np.array([[3.5, 5.3, -2.0],
              [-1.0, -0.5, 0.0]])

belief = np.array([[2, 5, 0],
                   [2, 0, 5]])

for i in range(3):
    x = belief[0, 0]
    sigw = 0.5*(5-x)**2 + 0.01
    sigma_w = np.array([[sigw, 0],
                        [0, sigw]])
    belief = kalman_ld(belief, sigma_v, sigma_w, U[:, i], Z[:, i])
print(belief)
```

```
[[-1.88251711  0.04818758  0.        ]
 [-0.03675163  0.          0.04818758]]
```