Void CV\_filter::Initialize\_Filter\_state\_covarience(float x,float y,float z,float vx,float vy,float vz,float time)

{

    int i,j,k,l;

    Sf.put\_Element(x,0,0)

    Sf.put\_Element(y,1,0)

    Sf.put\_Element(z,2,0)

    Sf.put\_Element(vx,3,0)

    Sf.put\_Element(vy,4,0)

    Sf.put\_Element(vz,5,0)

    this->Filtered\_Time=time;

    for(i=0;i<6;i++)

    {

        for(j=0;j<6:j++)

        {

            k=i%3;

            l=j%3;

            pf.put\_Element(R.Get\_Element(K,l),i,j);

        }

    }

}

Void CV\_filter::predict\_state\_covarience(float\_det)

{

Float T\_3,T\_2

Phi.put\_Element(1,0,0);

Phi.put\_Element(delt,0,3)

Phi.put\_Element(1,1,1)

Phi.put\_Element(delt,1,4)

Phi.put\_Element(1,2,2)

Phi.put\_Element(delt,2,5)

Phi.put\_Element(1,3,3)

Phi.put\_Element(1,4,4)

Phi.put\_Element(1,5,5)

Sp=Phi\*sf;

predicted\_Time=Filtered\_time+delt;

T\_3=(delt\*delt\*delt)/3.0;

T\_2=(delt\*delt)/2.0;

Q.put\_Element(T\_3,0,0);

Q.put\_Element(T\_3,1,1);

Q.put\_Element(T\_3,2,2);

Q.put\_Element(T\_2,0,3);

Q.put\_Element(T\_2,1,4);

Q.put\_Element(T\_2,2,5);

Q.put\_Element(T\_2,3,0);

Q.put\_Element(T\_2,4,1);

Q.put\_Element(T\_2,5,2);

Q.put\_Element(delt,3,3);

Q.put\_Element(delt,4,4);

Q.put\_Element(delt,5,5);

Q=Q\*plant\_noise;

Pp=Phi\*pf\*(Phi.matrix\_transpose())+Q

}

Void CV\_filter::Filter\_state\_covarience()

{

    Prev\_Sf=Sf;

    Prev\_Filtered.Time=Filtered\_time;

    S=R+H\*Pp\*(H.matrix\_Transpose());

    K=(Pp\*(H.matrix\_Transpose()))\*(S.matrix\_inverse());

    Inn=(Z-H\*Sp);

    Sf=Sp+K\*Inn;

    pf=(Inn-K\*H)\*Pp;

    Filtered\_Time=Meas\_Time;

}

sig\_e\_sqr=

#initializing R Matrix

rpt.R.d\_00=sig\_r\*sig\_r\*cos(e)\*cos(e)\*sin(a)\*sin(a)+r\*r\*cos(e)\*cos(e)\*cos(a)\*cos(a)+sig\_a\*sig\_a+r\*r\*sin(e)\*sin(e)\*sin(a)\*sin(a)\*sig\_e\_sqr;

rpt.R.d\_11=sig\_r\*sig\_r\*cos(e)\*cos(e)\*cos(a)\*cos(a)+r\*r\*cos(e)\*cos(e)\*sin(a)\*sin(a)+sig\_a\*sig\_a+r\*r\*sin(e)\*sin(e)\*cos(a)\*cos(a)\*sig\_e\_sqr;

rpt.R.d\_22=sig\_r\*sig\_r\*cos(e)\*cos(e)\*sin(a)\*sin(a)+r\*r\*cos(e)\*cos(e)\*cos(a)\*cos(a)+sig\_a\*sig\_a+r\*r\*sin(e)\*sin(e)\*sin(a)\*sin(a)\*sig\_e\_sqr;

import numpy as np

class CV\_filter:

def \_\_init\_\_(self):

# Initialize necessary attributes

pass

def Initialize\_Filter\_state\_covariance(self, x, y, z, vx, vy, vz, time, sig\_r, sig\_a, sig\_e\_sqr):

# Initialize state and covariance matrices

self.Sf = np.zeros((6, 1))

self.Sf[0] = x

self.Sf[1] = y

self.Sf[2] = z

self.Sf[3] = vx

self.Sf[4] = vy

self.Sf[5] = vz

self.Filtered\_Time = time

# Initialize covariance matrix R

self.R = np.zeros((3, 3))

self.R[0, 0] = sig\_r \* sig\_r \* np.cos(e) \* np.cos(e) \* np.sin(a) \* np.sin(a) + r \* r \* np.cos(e) \* np.cos(e) \* np.cos(a) \* np.cos(a) + sig\_a \* sig\_a + r \* r \* np.sin(e) \* np.sin(e) \* np.sin(a) \* np.sin(a) \* sig\_e\_sqr

self.R[1, 1] = sig\_r \* sig\_r \* np.cos(e) \* np.cos(e) \* np.cos(a) \* np.cos(a) + r \* r \* np.cos(e) \* np.cos(e) \* np.sin(a) \* np.sin(a) + sig\_a \* sig\_a + r \* r \* np.sin(e) \* np.sin(e) \* np.cos(a) \* np.cos(a) \* sig\_e\_sqr

self.R[2, 2] = sig\_r \* sig\_r \* np.cos(e) \* np.cos(e) \* np.sin(a) \* np.sin(a) + r \* r \* np.cos(e) \* np.cos(e) \* np.cos(a) \* np.cos(a) + sig\_a \* sig\_a + r \* r \* np.sin(e) \* np.sin(e) \* np.sin(a) \* np.sin(a) \* sig\_e\_sqr

# Initialize pf matrix (if needed)

self.pf = np.zeros((6, 6))

for i in range(6):

for j in range(6):

k = i % 3

l = j % 3

self.pf[i, j] = self.R[k, l]

def predict\_state\_covariance(self, delt, plant\_noise):

# Predict state covariance

self.Phi = np.eye(6)

self.Phi[0, 3] = delt

self.Phi[1, 4] = delt

self.Phi[2, 5] = delt

self.Sp = np.dot(self.Phi, self.Sf)

self.predicted\_Time = self.Filtered\_Time + delt

T\_3 = (delt \* delt \* delt) / 3.0

T\_2 = (delt \* delt) / 2.0

self.Q = np.zeros((6, 6))

self.Q[0, 0] = T\_3

self.Q[1, 1] = T\_3

self.Q[2, 2] = T\_3

self.Q[0, 3] = T\_2

self.Q[1, 4] = T\_2

self.Q[2, 5] = T\_2

self.Q[3, 0] = T\_2

self.Q[4, 1] = T\_2

self.Q[5, 2] = T\_2

self.Q[3, 3] = delt

self.Q[4, 4] = delt

self.Q[5, 5] = delt

self.Q = np.dot(self.Q, plant\_noise)

self.Pp = np.dot(np.dot(self.Phi, self.pf), self.Phi.T) + self.Q

def Filter\_state\_covariance(self, H, Z):

# Filter state covariance

self.Prev\_Sf = self.Sf.copy()

self.Prev\_Filtered\_Time = self.Filtered\_Time

self.S = self.R + np.dot(np.dot(H, self.Pp), H.T)

self.K = np.dot(np.dot(self.Pp, H.T), np.linalg.inv(self.S))

self.Inn = Z - np.dot(H, self.Sp)

self.Sf = self.Sp + np.dot(self.K, self.Inn)

self.pf = np.dot((self.Inn - np.dot(self.K, H)), self.Pp)

self.Filtered\_Time = self.Meas\_Time