Homework Assignment 3- Implementation
Arshad Nowsath(nowsath)
5-May-2020

1. Coordinated Turn Motion

```
function [fx, Fx] = coordinatedTurnMotion(x, T)
%COORDINATEDTURNMOTION calculates the predicted state using a coordinated
%turn motion model, and also calculated the motion model Jacobian
%
%Input:
%   x           [5 x 1] state vector
%   T           [1 x 1] Sampling time
%
%Output:
%   fx          [5 x 1] motion model evaluated at state x
%   Fx          [5 x 5] motion model Jacobian evaluated at state x
%
% NOTE: the motion model assumes that the state vector x consist of the
% following states:
%   px          X-position
%   py          Y-position
%   v           velocity
%   phi         heading
%   omega       turn-rate
% Your code for the motion model here
fx = [x(1,1)+T*x(3,1)*cos(x(4,1));
    x(2,1)+T*x(3,1)*sin(x(4,1));
    x(3,1);
    x(4,1)+T*x(5,1);
    x(5,1)];
%Check if the Jacobian is requested by the calling function
if nargout > 1
    % Your code for the motion model Jacobian here
    Fx = [1 0 T*cos(x(4,1)) -T*x(3,1)*sin(x(4,1)) 0;
        0 1 T*sin(x(4,1)) T*x(3,1)*cos(x(4,1)) 0;
        0 0 1 0 0;
        0 0 0 1 T;
        0 0 0 0 1];
end
end
```

2. Dual Bearing Measurement

```
function [hx, Hx] = dualBearingMeasurement(x, s1, s2)
%DUOBEARINGMEASUREMENT calculates the bearings from two sensors, located in
%s1 and s2, to the position given by the state vector x. Also returns the
%Jacobian of the model at x.
%
%Input:
%   x           [n x 1] State vector, the two first element are 2D position
%   s1          [2 x 1] Sensor position (2D) for sensor 1
%   s2          [2 x 1] Sensor position (2D) for sensor 2
%
%Output:
%   hx          [2 x 1] measurement vector
%   Hx          [2 x n] measurement model Jacobian
%
% NOTE: the measurement model assumes that in the state vector x, the first
% two states are X-position and Y-position.

% Your code here
Hx = zeros(2,size(x,1));
hx = [atan2(x(2)-s1(2) , x(1)-s1(1)) ; atan2(x(2)-s2(2) , x(1)-s2(1))];
dens1 = (x(1)-s1(1))^2 + (x(2)-s1(2))^2; dens2 = (x(1)-s2(1))^2 + (x(2)-s2(2))^2;
Hx(1:2,1:2) = [ (-x(2)+s1(2))/dens1, (x(1)-s1(1))/dens1; (-x(2)+s2(2))/dens2, (x(1)-
s2(1))/dens2];

end
```

3. Generate Non-Linear state sequence

```
function X = genNonLinearStateSequence(x_0, P_0, f, Q, N)
%GENNONLINEARSTATESEQUENCE generates an N+1-long sequence of states using a
%    Gaussian prior and a nonlinear Gaussian process model
%
%Input:
%   x_0         [n x 1] Prior mean
%   P_0         [n x n] Prior covariance
%   f           Motion model function handle
%               [fx,Fx]=f(x)
%               Takes as input x (state),
%               Returns fx and Fx, motion model and Jacobian evaluated at x
%               All other model parameters, such as sample time T,
%               must be included in the function
```

```
%   Q          [n x n] Process noise covariance
%   N          [1 x 1] Number of states to generate
%
%Output:
%   X          [n x N+1] State vector sequence
%


% Your code here
%determine the lentght of the state vector and allocate space for it
n=length(x_0);
X=zeros(n,N+1);


%then samples for the process noise are done and the initial state is
%determined using the gaussian inputs for the prior
q=mvnrnd(zeros(n,1),Q,N+1)';
X(:,1)=mvnrnd(x_0,P_0);


%using the linear gaussian process model the state sequence is generated
for i=2:N+1
    X(:,i)=f(X(:,i-1))+q(:,i-1);
end
end
```

4. Generate Non-Linear Measurement Sequence

```
function Y = genNonLinearMeasurementSequence(X, h, R)
%GENNONLINEARMEASUREMENTSEQUENCE generates ovservations of the states
% sequence X using a non-linear measurement model.
%
%Input:
%   X          [n x N+1] State vector sequence
%   h          Measurement model function handle
%   h          Measurement model function handle
%              [hx,Hx]=h(x)
%              Takes as input x (state)
%              Returns hx and Hx, measurement model and Jacobian evaluated at x
%   R          [m x m] Measurement noise covariance
%
%Output:
%   Y          [m x N] Measurement sequence
% Your code here
%Size of sequence vector
```

```
M=length(R);N=size(X,2)-1;


%space for measurement vector is allocated
Y=zeros(M,N);


%Sampling
r=mvnrnd(zeros(M,1),R,N)';


%generate sequence
for i=1:N
    Y(:,i)=h(X(:,i+1))+r(:,i);
end
end
```

5. Sigma Points

```
function [SP,W] = sigmaPoints(x, P, type)
% SIGMAPOINTS computes sigma points, either using unscented transform or
% using cubature.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%
%Output:
%   SP          [n x 2n+1] UKF, [n x 2n] CKF. Matrix with sigma points
%   W           [1 x 2n+1] UKF, [1 x 2n] UKF. Vector with sigma point weights
%
        P_d2=sqrtm(P);
        n=length(x);
    switch type
        case 'UKF'
            w0=1-n/3;
            wi=(1-w0)/(2*n);
            wp=sqrt(n/(1-w0));

            W=ones(1,2*n+1)*wi;
            W(1)=w0;

            for i=0:1:n
                if i==0
```

```
            SP(:,i+1)=x;
        else
            SP(:,i+1)=x+wp*P_d2(:,i);
            SP(:,i+1+n)=x-wp*P_d2(:,i);
        end
    end

case 'CKF'
    wi=1/(2*n);
    wp=sqrt(n);

    W=ones(1,2*n)*wi;

    for i=1:1:n
        SP(:,i)=x+wp*P_d2(:,i);
        SP(:,i+n)=x-wp*P_d2(:,i);
    end
otherwise
    error('Incorrect type of sigma point')
end
end
```

6. Non-Linear Kalman Filter Prediction

```
unction [x, P] = nonLinKFprediction(x, P, f, Q, type)
%NONLINKFPREDICTION calculates mean and covariance of predicted state
%   density using a non-linear Gaussian model.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%   f           Motion model function handle
%               [fx,Fx]=f(x)
%               Takes as input x (state),
%               Returns fx and Fx, motion model and Jacobian evaluated at x
%               All other model parameters, such as sample time T,
%               must be included in the function
%   Q           [n x n] Process noise covariance
%   type        String that specifies the type of non-linear filter
%
%Output:
%   x           [n x 1] predicted state mean
```

```
%   P          [n x n] predicted state covariance
%


   [fx,Fx]=f(x);
   n=length(x);
   switch type
      case 'EKF'
         x=fx;
         P=Fx*P*Fx'+Q;


      case 'UKF'
         [SP,W] = sigmaPoints(x, P, 'UKF');

         for i=0:2*n
            [fx,Fx]=f(SP(:,i+1));
            x(:,i+1)=fx*W(:,i+1);
         end
         x=sum(x,2);

         for i=0:2*n
            [fx,Fx]=f(SP(:,i+1));
            P(:,:,i+1)=(fx-x)*(fx-x)'*W(:,i+1);
         end
         P=sum(P,3)+Q;

         % Make sure the covariance matrix is semi-definite
         if min(eig(P))<=0
            [v,e] = eig(P, 'vector');
            e(e<0) = 1e-4;
            P = v*diag(e)/v;
         end

      case 'CKF'
         [SP,W] = sigmaPoints(x, P, 'CKF');

         for i=1:2*n
            [fx,Fx]=f(SP(:,i));
            x(:,i)=fx*W(:,i);
         end
         x=sum(x,2);

         for i=1:2*n
            [fx,Fx]=f(SP(:,i));
```

```
            P(:,:,i)=(fx-x)*(fx-x)'*W(:,i);
        end
        P=sum(P,3)+Q;
    otherwise
        error('Incorrect type of non-linear Kalman filter')
end


end
```

7. Non Linear Kalman Filter Update

```
function [x, P] = nonLinKFupdate(x, P, y, h, R, type)
%NONLINKFUPDATE calculates mean and covariance of predicted state
%   density using a non-linear Gaussian model.
%
%Input:
%   x           [n x 1] Prior mean
%   P           [n x n] Prior covariance
%   y           [m x 1] measurement vector
%   h           Measurement model function handle
%               [hx,Hx]=h(x)
%               Takes as input x (state),
%               Returns hx and Hx, measurement model and Jacobian evaluated at x
%               Function must include all model parameters for the particular model,
%               such as sensor position for some models.
%   R           [m x m] Measurement noise covariance
%   type        String that specifies the type of non-linear filter
%
%Output:
%   x           [n x 1] updated state mean
%   P           [n x n] updated state covariance
%
    [hx,Hx]=h(x);
    n=length(x);
    m=length(y);
    switch type
        case 'EKF'
            S=Hx*P*Hx'+R;
            K=P*Hx'*S^-1;

            x=x+K*(y-hx);
            P=P-K*S*K';
        case 'UKF'
```

```
    [SP,W]=sigmaPoints(x,P,'UKF');

    for i=0:2*n
        yp(:,i+1)=h(SP(:,i+1))*W(i+1);
    end
    yp=sum(yp,2);

    for i=0:2*n
        Pp(:,:,i+1)=(SP(:,i+1)-x)*(h(SP(:,i+1))-yp)'*W(i+1);
    end
    Pp=sum(Pp,3);

    for i=0:2*n
        Sp(:,:,i+1)=(h(SP(:,i+1))-yp)*(h(SP(:,i+1))-yp)'*W(i+1);
    end
    Sp=sum(Sp,3)+R;

    x=x+Pp*inv(Sp)*(y-yp);
    P=P-Pp*inv(Sp)*Pp';

    % Make sure the covariance matrix is semi-definite
    if min(eig(P))<=0
        [v,e] = eig(P, 'vector');
        e(e<0) = 1e-4;
        P = v*diag(e)/v;
    end

case 'CKF'
    [SP,W]=sigmaPoints(x,P,'CKF');

    for i=1:2*n
        yp(:,i)=h(SP(:,i))*W(i);
    end
    yp=sum(yp,2);

    for i=1:2*n
        Pp(:,:,i)=(SP(:,i)-x)*(h(SP(:,i))-yp)'*W(i);
    end
    Pp=sum(Pp,3);

    for i=1:2*n
        Sp(:,:,i)=(h(SP(:,i))-yp)*(h(SP(:,i))-yp)'*W(i);
    end
    Sp=sum(Sp,3)+R;
```

```
        x=x+Pp*Sp^-1*(y-yp);
        P=P-Pp*Sp^-1*Pp';

    otherwise
        error('Incorrect type of non-linear Kalman filter')
  end


end
```

8. Non Linear Kalman Filter

```
function [xf, Pf, xp, Pp] = nonLinearKalmanFilter(Y, x_0, P_0, f, Q, h, R, type)
%NONLINEARKALMANFILTER Filters measurement sequence Y using a
% non-linear Kalman filter.
%
%Input:
%   Y           [m x N] Measurement sequence for times 1,...,N
%   x_0         [n x 1] Prior mean for time 0
%   P_0         [n x n] Prior covariance
%   f                Motion model function handle
%                    [fx,Fx]=f(x)
%                    Takes as input x (state)
%                    Returns fx and Fx, motion model and Jacobian evaluated at x
%   Q           [n x n] Process noise covariance
%   h                Measurement model function handle
%                    [hx,Hx]=h(x,T)
%                    Takes as input x (state),
%                    Returns hx and Hx, measurement model and Jacobian evaluated at x
%   R           [m x m] Measurement noise covariance
%
%Output:
%   xf          [n x N]     Filtered estimates for times 1,...,N
%   Pf          [n x n x N] Filter error convariance
%   xp          [n x N]     Predicted estimates for times 1,...,N
%   Pp          [n x n x N] Filter error convariance
%

% Your code here. If you have good code for the Kalman filter, you should re-use it here as
% much as possible.
%% Parameters
N = size(Y,2); n = length(x_0); m = size(Y,1);

%% Data allocation
xf = zeros(n,N+1); Pf = zeros(n,n,N+1);
xp = zeros(n,N); Pp = zeros(n,n,N);
```

```
%initial
xf(:,1) = x_0; Pf(:,:,1) = P_0;

%kalman
for i=2:N+1
    [xp(:,i-1), Pp(:,:,i-1)] = nonLinKFprediction(xf(:,i-1), Pf(:,:,i-1), f, Q, type);
    [xf(:,i), Pf(:,:,i)] = nonLinKFupdate(xp(:,i-1), Pp(:,:,i-1), Y(:,i-1), h, R, type);
end

%output
xf = xf(:,2:end); Pf = Pf(:,:,2:end);
end
```