Simulação em Matlab/Simulink

Grupo:

André Abido Figueiró / Tiago Bornia de Castro

Primeira Simulação:

Para demonstrar o funcionamento do Filtro de Kalman Estendido e Unscented, foi realizada uma simulação em Simulink, apresentada na Figura 1. O sistema (Planta) consiste num veículo movendo-se no plano, sendo as entradas a velocidade e o ângulo para o qual o veículo se move. São considerados ruídos de processo somados ao ângulo (com variância 0.1) e para a velocidade (variância = 1). Além disso, é somado um ruído de medição às medidas de posição X e Y (variância = 1).

O bloco "Cinemática" simula o comportamento do sistema ao longo do tempo, ao passo que o bloco "UKF" obtém as medidas ruidosas da posição e os sinais de referência (que corresponde às entradas de comando para o veículo) para estimar a posição real do veículo, a partir de uma condição inicial arbitrária. Os códigos das s-functions correspondentes a esses blocos estão no final deste documento.

Vale notar que não foi observada diferença relevante entre o UKF e EKF, indicando que as linearizações desses dois métodos obtiveram resultados semelhantes.

A Figura 2 apresenta a trajetória do veículo, sendo possível notar que a estimativa corrige rapidamente o erro inserido na estimativa inicial, além de rejeitar os ruídos de medição. Já a Figura 3 apresenta as variâncias estimadas para X e Y nas primeiras iterações, indicando o maior grau de certeza do estimador por UKF para a posição do veículo.

Tempo de Amostragem	1
Variância do ruído de mediçao	R = [1 0 ; 0 1]
Variância do ruído de processo	Q = [1 0 ; 0 0.1]

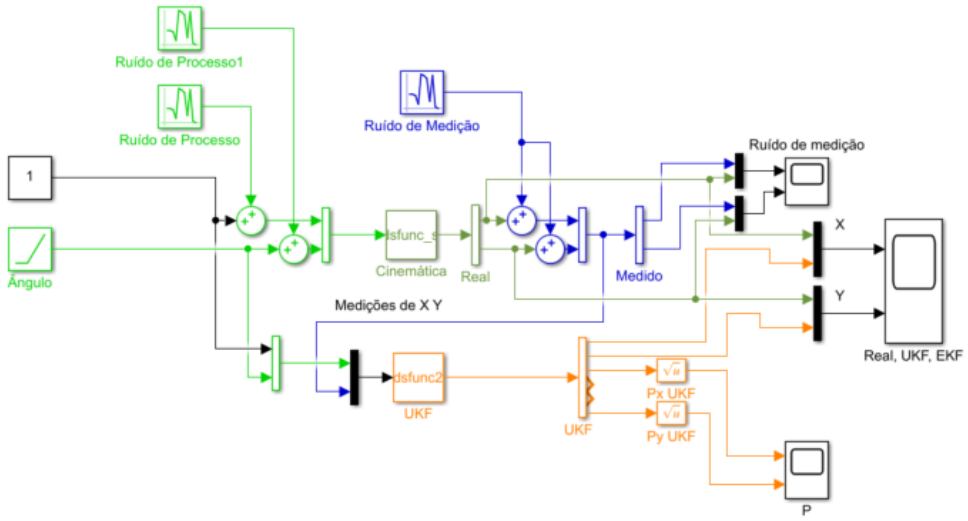

Figura 1 - Simulação realizada no Simulink

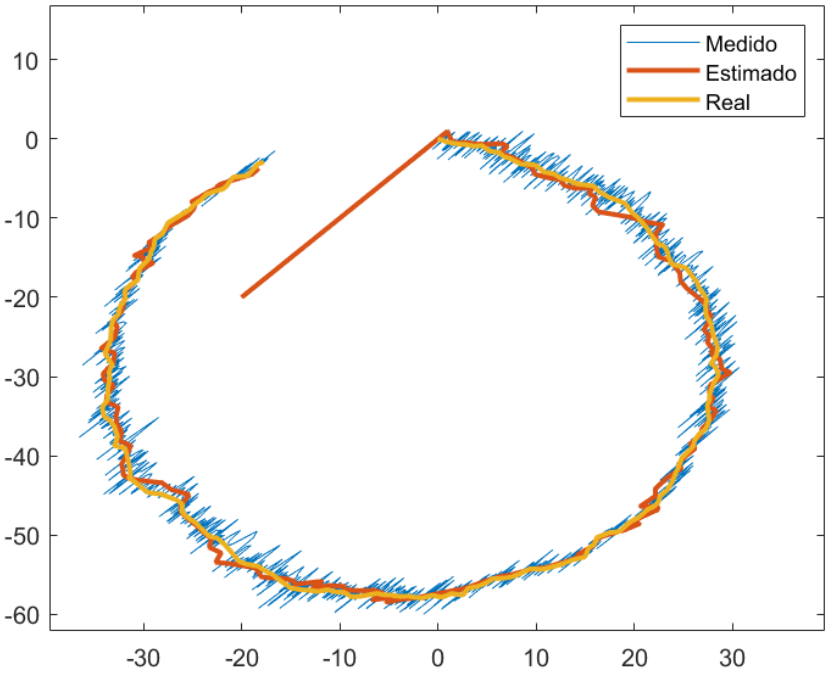

Figura 2 - Trajetória Medida, Estimada e Real do veículo.

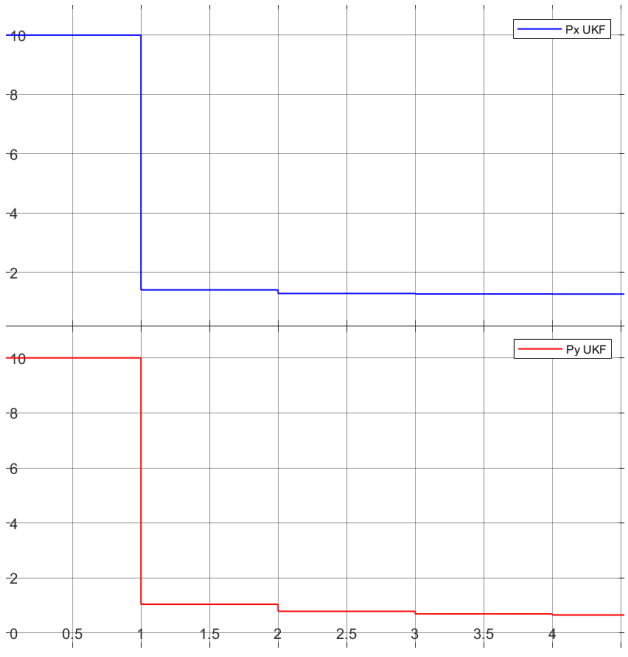

Figura 3 - Variância da posição X e Y do veículo estimada pelo UKF.

Segunda Simulação

Tendo em vista não haverem diferenças entre o EKF e o UKF na primeira simulação, buscou-se apresentar uma simulação que induza maior divergência entre os métodos de linearização. Utilizou-se, para isto, a medida dos estados do veículo em coordenadas polares. Esse modelo simula o rastreamento de um alvo por um observador externo com medida de ângulo e distância.

A Figura 4 apresenta montagem da simulação no Simulink, ao parro que a Figura 5 apresenta os resultados da estimação.

Tempo de Amostragem	2
Variância do ruído de mediçao	R=[100 0; 0 10*pi/360];
Variância do ruído de processo	Q = [1 0 ; 0 0.1]

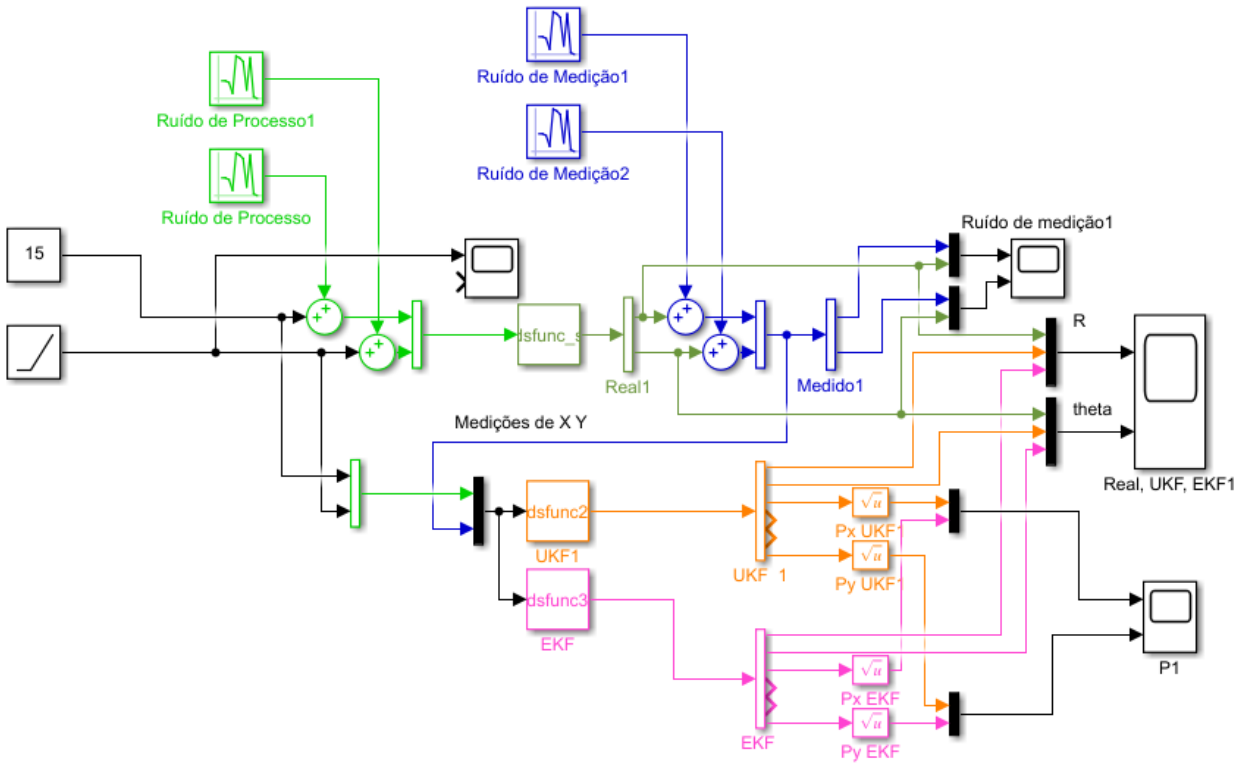

Figura 4 - Veiculo rastreado por coordenadas polares.

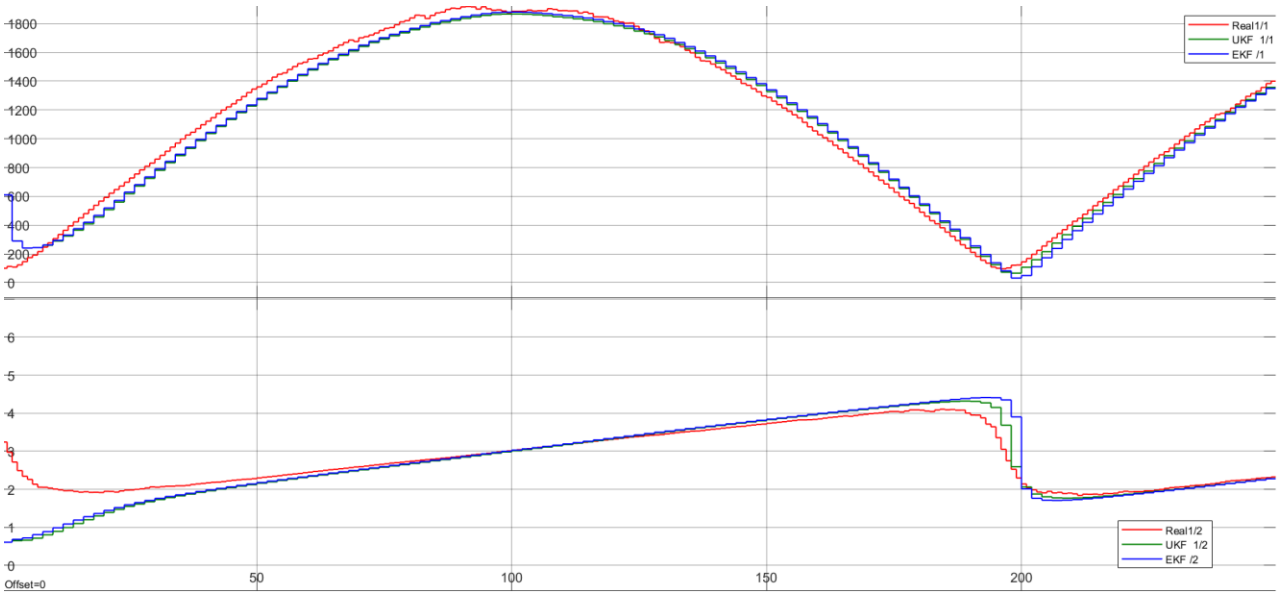

Figura 5 - Distância e ângulo real e estimado por UKF e EKF.

CINEMÁTICA:

```
function [sys,x0,str,ts] = dsfunc_s(t,x,u,flag, x0a)

t_s=1;

switch flag,

  case 0,
        sizes = simsizes;
        sizes.NumContStates  = 0;
        sizes.NumDiscStates  = 2;
        sizes.NumOutputs     = 2;
        sizes.NumInputs      = 2;
        sizes.DirFeedthrough = 1;
        sizes.NumSampleTimes = 1;
        sys = simsizes(sizes);
        x0 =[x0a ];
        str = [];
        ts  = [t_s 0];
  case 2,
  %
    f=@(x)[x(1)+cos(u(2))*(u(1))*t_s ; x(2)-sin(u(2))*(u(1))*t_s  ];
    sys =[f(x)'];
  case 3,
    r = sqrt(x(1)^2+x(2)^2);
    theta=atan2(x(1),x(2));
    sys = [r wrapTo2Pi(theta)];
  case 9,
    sys = []; % do nothing
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
End
```

ESTIMADOR:

```
function [sys,x0,str,ts] = dsfunc2(t,x,u,flag, x0a)

%r=1;
Q=[1 0; 0 0.1];
R=[100 0; 0 10*pi/360];
t_s=2;

switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,

        sizes = simsizes;
        sizes.NumContStates  = 0;
        sizes.NumDiscStates  = 6;
        sizes.NumOutputs     = 6;
        sizes.NumInputs      = 4;
        sizes.DirFeedthrough = 1;
        sizes.NumSampleTimes = 1;
        sys = simsizes(sizes);
        x0 =[x0a ];
        str = [];
        ts  = [t_s 0];
```

```
% end mdlInitializeSizes

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
  %
    P=[x(3) x(4); x(5) x(6)];
    f=@(x)[x(1)+cos(u(2))*(u(1))*t_s ; x(2)-sin(u(2))*(u(1))*t_s  ];
    z=u(3:4);
    h=@(x)[sqrt(x(1)^2+x(2)^2) angle(x(1)+1i*x(2))];
    [a,P]=ukf(f,[x(1:2)],P,h,z,Q,R);
    sys =[a(1) a(2) P(1,:) P(2,:)];
  %%%%%%%%%%
  % Output %
  %%%%%%%%%%
  case 3,
    r = sqrt(x(1)^2+x(2)^2);
    if abs(x(1)*x(2))>0
        theta=atan2(x(1),x(2));
    else
        theta=0;
    end
    sys = [r wrapTo2Pi(theta) x(3:6)'];

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
  case 9,
    sys = []; % do nothing

  %%%%%%%%%%%%%%%%%%%%
  % Unexpected flags %
  %%%%%%%%%%%%%%%%%%%%
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
end
```

UKF:

```
function [x,P]=ukf2(fstate,x,P,hmeas,z,Q,R)
% UKF   Unscented Kalman Filter for nonlinear dynamic systems
% [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance, P
% for nonlinear dynamic system (for simplicity, noises are assumed as additive):
%           x_k+1 = f(x_k) + w_k
%           z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%       v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:   f: function handle for f(x)
%           x: "a priori" state estimate
%           P: "a priori" estimated state covariance
%           h: fanction handle for h(x)
%           z: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance
%
% Example:
% Reference: Julier, SJ. and Uhlmann, J.K., Unscented Filtering and
% Nonlinear Estimation, Proceedings of the IEEE, Vol. 92, No. 3,
% pp.401-422, 2004.
%
% By Yi Cao at Cranfield University, 04/01/2008
%
```

```
L=numel(x);                                 %numer of states
m=numel(z);                                 %numer of measurements
alpha=1e-3;                                 %default, tunable
 ki=0;                                       %default, tunable
 beta=2;
lambda=0.5;
c=(L+lambda);
Wm=[lambda/c  0.5/c+zeros(1,2*L)];
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);               %weights for covariance


c=sqrt(c);
X=sigmas(x,P,c);                            %sigma points around x
[x1,X1,P1,X2]=ut(fstate,X,Wm,Wc,L,Q);          %unscented transformation of process
% X1=sigmas(x1,P1,c);                         %sigma points around x1
% X2=X1-x1(:,ones(1,size(X1,2)));             %deviation of X1
[z1,Z1,P2,Z2]=ut(hmeas,X1,Wm,Wc,m,R);       %unscented transformation of measurments
P12=X2*diag(Wc)*Z2';                        %transformed cross-covariance
K=P12*inv(P2);
x=x1+(K)*(z-z1);                            %state update
P=P1-(K)*P12';                              %covariance update

function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
%Unscented Transformation
%Input:
%        f: nonlinear map
%        X: sigma points
%       Wm: weights for mean
%       Wc: weights for covraiance
%        n: numer of outputs of f
%        R: additive covariance
%Output:
%        y: transformed mean
%        Y: transformed smapling points
%        P: transformed covariance
%       Y1: transformed deviations

L=size(X,2);
y=zeros(n,1);
Y=zeros(n,L);
for k=1:L
    Y(:,k)=f(X(:,k));
    y=y+Wm(k)*Y(:,k);
end
Y1=Y-y(:,ones(1,L));
P=Y1*diag(Wc)*Y1'+R;



function X=sigmas(x,P,c)
%Sigma points around reference point
%Inputs:
%       x: reference point
%       P: covariance
%       c: coefficient
%Output:
%       X: Sigma points

A = c*chol(P)';
Y = x(:,ones(1,numel(x)));
X = [x Y+A Y-A];
```

EKF:

```
function [x,P]=ekf(fstate,x,P,hmeas,z,R,Q)
                            %Predição
[x1,A]=jaccsd(fstate,x);    %1 - Atualiza média, calcula Jacobiano de g(x)
P=A*P*A'+R;                 %2 - Atualiza  matr. covariância
                            %Correção
[z1,H]=jaccsd(hmeas,x1);    %3 - Calcula Jacobiano de h(x)
P12=P*H';                   %
K=P12/(H*P12+Q);            %4 - Ganho de Kalman
x=x1+K*(z-z1);              %5 - Corrige média
P=P-K*P12';                 %6 - Corrige matr. covariância

function [z,A]=jaccsd(fun,x)
% JACCSD Jacobian through complex step differentiation
% [z J] = jaccsd(f,x)
% z = f(x)
% J = f'(x)
%
z=fun(x);
n=numel(x);
m=numel(z);
A=zeros(m,n);
h=n*eps;
for k=1:n
    x1=x;
    x1(k)=x1(k)+h*1i;
    A(:,k)=imag(fun(x1))/h;
end
```