Nome: Fabio Grassiotto
RA: 890441
Disciplina: IA368N, 2º S 2018

Atividade 4 – Filtro de Kalman estendido baseado em linha para localização de robôs

Objetivo

O objetivo desta atividade é, dando continuidade à atividade anterior, mostrar através de um mapa conhecido das características lineares da representação das estruturas percebidas, que um robô pode se localizar. Para tanto será utilizado um filtro extendido de Kalman.

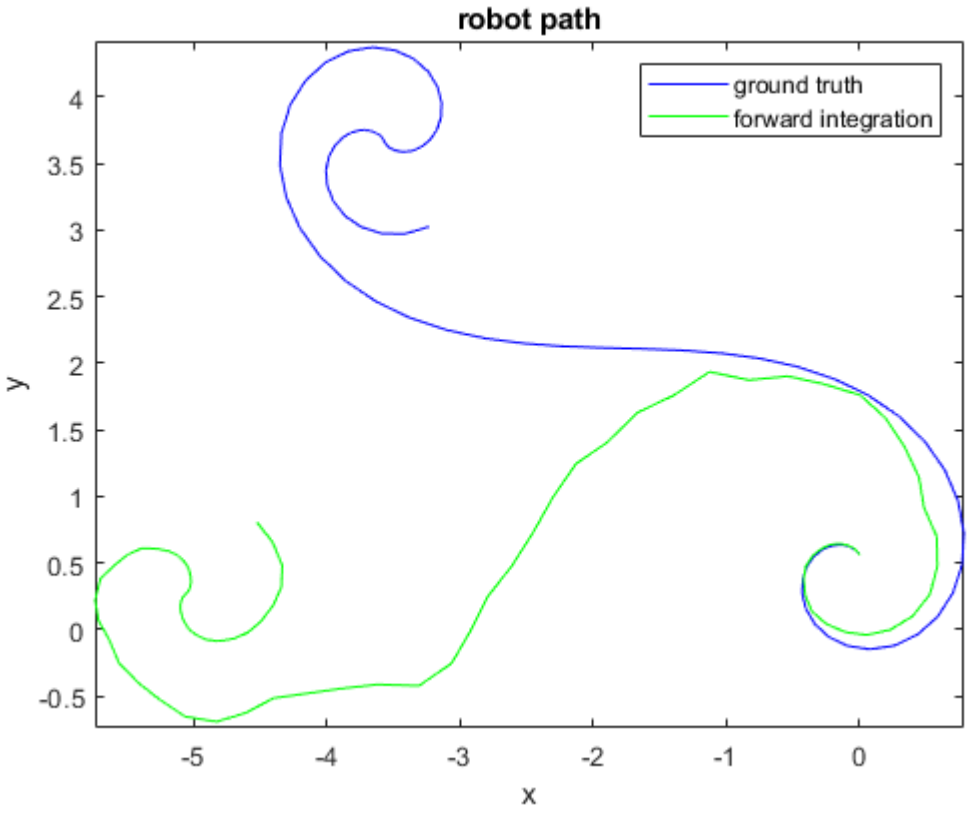
Tarefa 1

Nesta primeira tarefa, uma função *transitionFunction()* foi implementada para estimar o estado atual de um robô diferencial x_t baseado no estado anterior x_{t-1} assim como as entradas de controle u_t. A função também retorna os Jacobianos da função de transição de estado em relação ao estado atual e às entradas de controle.

```
1.  function [f, F_x, F_u] = transitionFunction(x,u, l)
2.  % [f, F_x, F_u] = transitionFunction(x,u,l) predicts the state x at time t given
3.  % the state at time t-1 and the input u at time t. F_x denotes the Jacobian
4.  % of the state transition function with respect to the state evaluated at
5.  % the state and input provided. F_u denotes the Jacobian of the state
6.  % transition function with respect to the input evaluated at the state and
7.  % input provided.
8.  % State and input are defined according to "Introduction to Autonomous Mobile Robo
    ts", pp. 337
9.
10. %STARTRM
11.
12. xX = x(1);
13. xY = x(2);
14. xTheta = x(3);
15. uDeltaSl = u(1);
16. uDeltaSr = u(2);
17.
18. % f is the estimate for the state x at T t.
19. f = [xX;xY;xTheta] + [ ((uDeltaSr + uDeltaSl)/2)*cos(xTheta + ((uDeltaSr-
    uDeltaSl)/2*l));
20.           ((uDeltaSl + uDeltaSr)/2)*sin(xTheta + ((uDeltaSr-uDeltaSl)/2*l));
21.           (uDeltaSr - uDeltaSl)/l];
22.
23. % Jacobians - formulas by hand.
24. F_x = [ 1 0 -
    (((uDeltaSr + uDeltaSl)/2)*sin(xTheta+(((uDeltaSr - uDeltaSl)/l)/2)));
25.         0 1 (((uDeltaSr + uDeltaSl)/2)*cos(xTheta+(((uDeltaSr - uDeltaSl)/l)/2)));

26.         0 0 1];
27.
28. F_u = [cos(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))/2 + (((uDeltaSr + uDeltaSl)/2)/(2
    *l))*sin(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))...
29.           cos(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))/2 - (((uDeltaSr + uDeltaSl)/2)
    /(2*l))*sin(xTheta+(((uDeltaSr - uDeltaSl)/l)/2));
```

```
30.         sin(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))/2 - (((uDeltaSr + uDeltaSl)/2)/(2
    *l))*cos(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))...
31.            sin(xTheta+(((uDeltaSr - uDeltaSl)/l)/2))/2 + (((uDeltaSr + uDeltaSl)/2)
    /(2*l))*cos(xTheta+(((uDeltaSr - uDeltaSl)/l)/2));
32.         (-1/(l))...
33.            (1/(l))] ;
34.
35.
36. %ENDRM
```

O código foi validado utilizando a função *validateTransitionFunction()*. Notei que a tolerância entre o resultado esperado e a saída da função *transitionfunction()* precisou ser alterada de 10^{-5} para 10^{-3}, obtendo o gráfico abaixo:



Tarefa 2

De forma similar à tarefa 1, partimos para a etapa de atualização da posição do robô, implementando a função *measurementFunction():*

```
1.  function [h, H_x] = measurementFunction(x, m)
2.  % [h, H_x] = measurementFunction(x, m) returns the predicted measurement
3.  % given a state x and a single map entry m. H_x denotes the Jacobian of the
4.  % measurement function with respect to the state evaluated at the state
5.  % provided.
6.  % Map entry and state are defined according to "Introduction to Autonomous Mobile
    Robots" pp. 337
7.
8.  %STARTRM
9.
10. % First we need to compute the the transformation from a line expressed in the wor
    ld coordinate
```

```
% frame into the body coordinate frame of the robot, z^t, given by the
% return parameter h. m(1) and m(2) are map entries.
% This formula is taken from the reference (1), 5.94.
xX = x(1);
xY = x(2);
xTheta = x(3);
Map1 = m(1);
Map2 = m(2);

h = [Map1 - xTheta;
     Map2 - (xX*cos(Map1) + xY*sin(Map1))];

% Jacobian of h above. (5.95).
H_x = [0          0          -1;
       -cos(Map1) -sin(Map1)  0];
%ENDRM

[h(1), h(2), isRNegated] = normalizeLineParameters(h(1), h(2));

if isRNegated
    H_x(2, :) = - H_x(2, :);
end
```

Foi verificado que o modelamento da medição percebida pelo robô obteve um resultado correto ao executar a função *validateMeasurementFunction()*.

```
>> validateMeasurementFunction
measurement function appears to be correct!
```

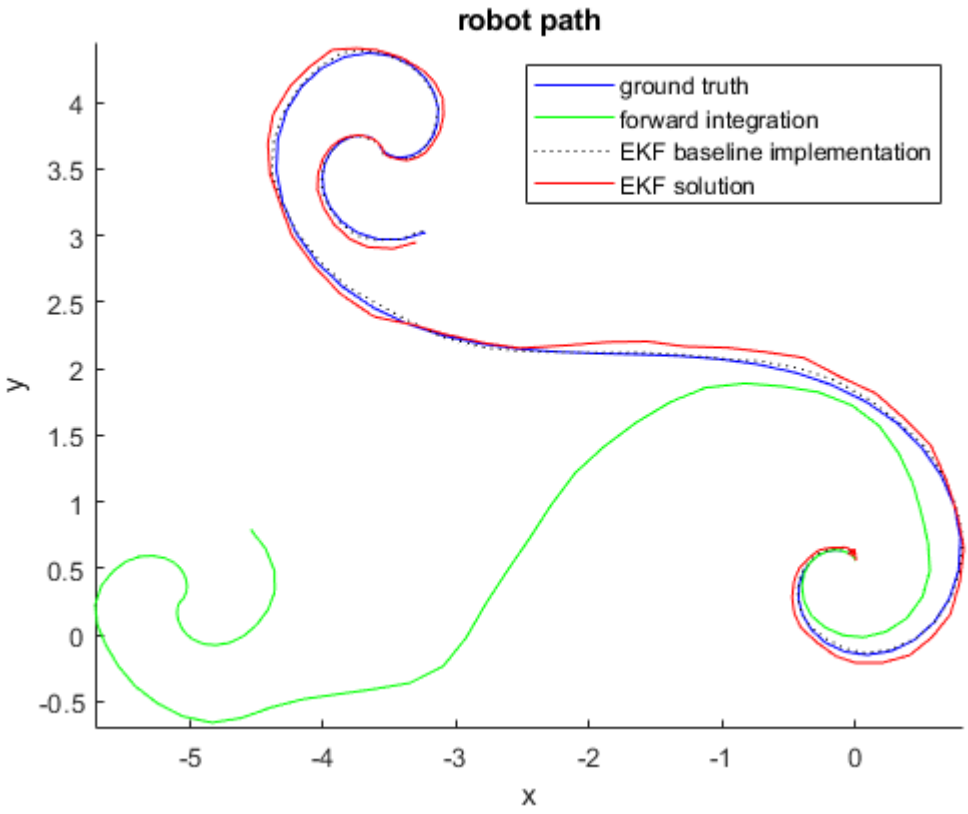
Tarefa 3

Nesta atividade foi implementada uma função *filterStep()* para executar as atualizações do filtro de Kalman usando para tanto os resultados obtidos nas tarefas anteriores. Foram utilizadas as funções *reshape* e *permute* do Matlab para manipulação e formatação dos resultados obtidos.

```
function [x_posteriori, P_posteriori] = filterStep(x, P, u, Z, R, M, k, g, l)
% [x_posteriori, P_posteriori] = filterStep(x, P, u, z, R, M, k, g, l)
% returns an a posteriori estimate of the state and its covariance

%STARTRM

uDeltaSl = u(1);
uDeltaSr = u(2);

% Q below is the Covariance of the noise associated to the motion model.
% propagate the state (p. 338) , here kr=kl=k

Q = [k*abs(uDeltaSr) 0;
     0 k*abs(uDeltaSl)];

% From 1st activity
[x_priori, F_x, F_u] = transitionFunction(x,u,l);
P_priori = (F_x*P*(F_x')) + (F_u*Q*(F_u'));

if size(Z,2) == 0
    x_posteriori = x_priori;
    P_posteriori = P_priori;
    return;
end

[v, H, R] = associateMeasurements(x_priori, P_priori, Z, R, M, g);
```

```
27.
28. y = reshape(v, [], 1);
29. H = reshape(permute(H, [1,3,2]), [], 3);
30. R = blockDiagonal(R);
31.
32. % update state estimates (pp. 335)
33. S = (H*P_priori*(H'))+R;
34. K = P_priori*(H')*(inv(S));
35.
36. x_posteriori = x_priori + K * y;
37. P_posteriori = (eye(size(P_priori)) - K*H) * P_priori;
38.
39. %ENDRM
```

Validando-se os dados extraídos na função acima, obtemos no gráfico abaixo a saída do filtro de Kalman e a integração das entradas de controle:



Tarefa 4

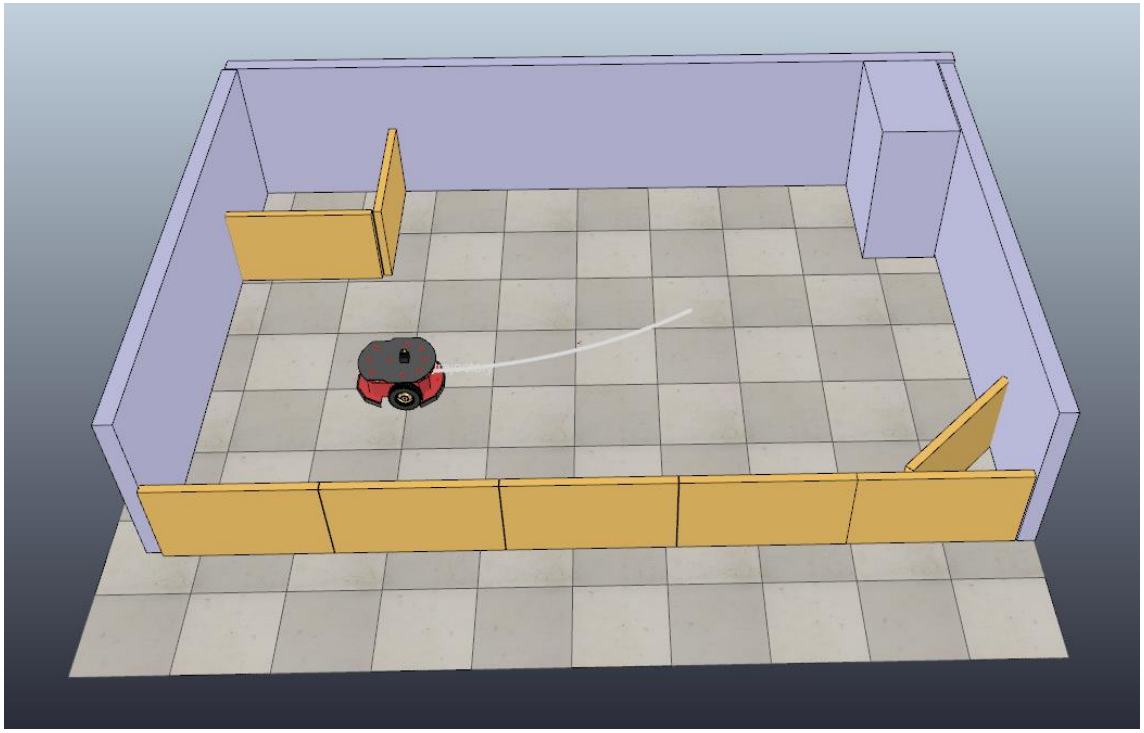
Nesta tarefa final, as atividades de extração de linhas e localização utilizando o filtro de Kalman serão combinadas, implementando uma solução completa e integrando no ambiente de simulação V-Rep.

Para tanto, uma função *incrementalLocalization()* foi implementada para retornar uma estimativa *a posteriori* da posição do robô assim como sua covariância.

```
1. function [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S, M, param
   s, k, g, l)
2. % [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S, R, M,
3. % k, l, g) returns the a posterori estimate of the state and its covariance,
```

```
% given the previous state estimate, control inputs, laser measurements and
% the map

C_TR = diag([repmat(0.1^2, 1, size(S, 2)) repmat(0.1^2, 1, size(S, 2))]);
[z, R, ans] = extractLinesPolar(S(1,:), S(2,:), C_TR, params);


%STARTRM
figure(2), cla, hold on;

%compute z_prior
z_prior=[];
% Now we need to loop through the map.
nM = size(M, 2); % number of map entries.

% Loop through all Map Entries
for i=1:nM
    % Call measurement function for each set of map entries.
    [h, ~]= measurementFunction(x, M(:,i));
    z_prior = [z_prior; h(1) h(2)];
end
% Just rearrange data at the end for plotting data.
z_prior = z_prior';

plot(z(1,:), z(2,:),'bo');
plot(z_prior(1,:), z_prior(2,:),'rx');
xlabel('angle [rad]'); ylabel('distance [m]')
legend('measurement','prior')
drawnow

% estimate robot pose
% Use the function from the previous step here.
[x_posterori, P_posterori] = filterStep(x, P, u, z, R, M, k, g, l);

%ENDRM
```

Validando os resultados obtidos, pode-se verificar no V-Rep a trajetória de movimentação circular do robô P3-Dx:



Conclusão

Os exercícios propostos permitiram exercitar a implementação de um filtro de Kalman estendido, através dos passos de predição e atualização (tarefas 1 e 2), até obtermos a saída do filtro (tarefa 3) e verificarmos a aplicação em um ambiente de simulação (tarefa 4).

Referências

(1) Roland Siegwart , Illah R. Nourbakhsh , Davide Scaramuzza, Introduction to Autonomous Mobile Robots, The MIT Press, 2011