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.

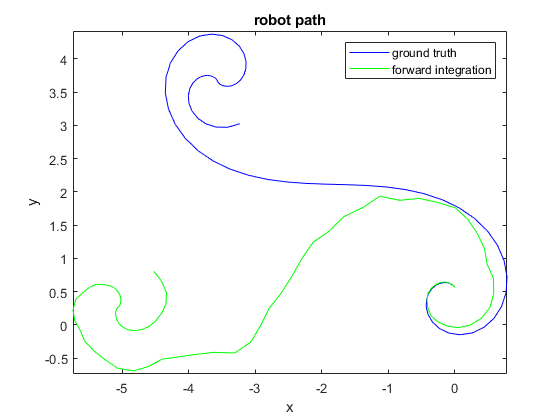
**Tarefa 1**

Nesta primeira tarefa, uma função *transitionFunction()* foi implementada para estimar o estado atual de um robô diferencial xt baseado no estado anterior xt-1 assim como as entradas de controle ut. 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 Robots", pp. 337
10. %STARTRM
12. xX = x(1);
13. xY = x(2);
14. xTheta = x(3);
15. uDeltaSl = u(1);
16. uDeltaSr = u(2);
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];
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];
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))] ;

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
8. %STARTRM
10. % First we need to compute the the transformation from a line expressed in the world coordinate
11. % frame into the body coordinate frame of the robot, z^t, given by the
12. % **return** parameter h. m(1) and m(2) are map entries.
13. % This formula is taken from the reference (1), 5.94.
14. xX = x(1);
15. xY = x(2);
16. xTheta = x(3);
17. Map1 = m(1);
18. Map2 = m(2);
20. h = [Map1 - xTheta;
21. Map2 - (xX\*cos(Map1) + xY\*sin(Map1))];
23. % Jacobian of h above. (5.95).
24. H\_x = [0          0          -1;
25. -cos(Map1) -sin(Map1)  0];
26. %ENDRM
28. [h(1), h(2), isRNegated] = normalizeLineParameters(h(1), h(2));
30. **if** isRNegated
31. H\_x(2, :) = - H\_x(2, :);
32. end

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

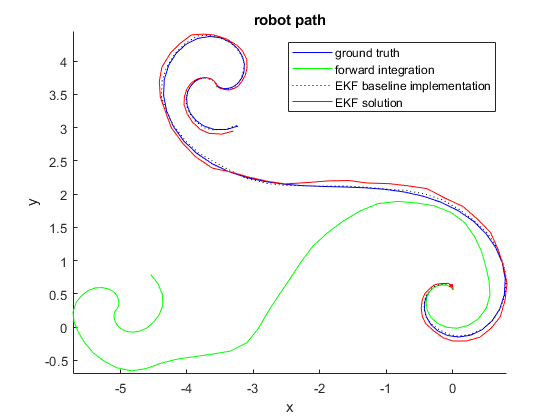
1. >> validateMeasurementFunction
2. measurement function appears to be correct!

**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.

1. function [x\_posteriori, P\_posteriori] = filterStep(x, P, u, Z, R, M, k, g, l)
2. % [x\_posteriori, P\_posteriori] = filterStep(x, P, u, z, R, M, k, g, l)
3. % returns an a posteriori estimate of the state and its covariance
5. %STARTRM
7. uDeltaSl = u(1);
8. uDeltaSr = u(2);
10. % Q below is the Covariance of the noise associated to the motion model.
11. % propagate the state (p. 338) , here kr=kl=k
13. Q = [k\*abs(uDeltaSr) 0;
14. 0 k\*abs(uDeltaSl)];
16. % From 1st activity
17. [x\_priori, F\_x, F\_u] = transitionFunction(x,u,l);
18. P\_priori = (F\_x\*P\*(F\_x')) + (F\_u\*Q\*(F\_u'));
20. **if** size(Z,2) == 0
21. x\_posteriori = x\_priori;
22. P\_posteriori = P\_priori;
23. **return**;
24. end
26. [v, H, R] = associateMeasurements(x\_priori, P\_priori, Z, R, M, g);
28. y = reshape(v, [], 1);
29. H = reshape(permute(H, [1,3,2]), [], 3);
30. R = blockDiagonal(R);
32. % update state estimates (pp. 335)
33. S = (H\*P\_priori\*(H'))+R;
34. K = P\_priori\*(H')\*(inv(S));
36. x\_posteriori = x\_priori + K \* y;
37. P\_posteriori = (eye(size(P\_priori)) - K\*H) \* P\_priori;
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**

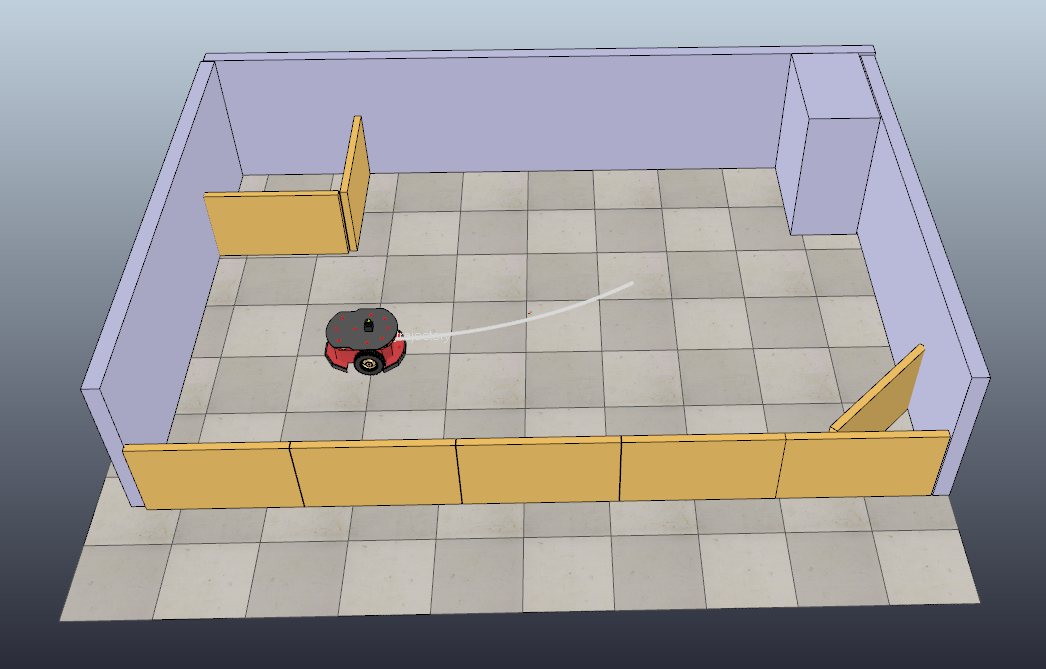
Nesta tarefa final, as atividades de extração de linhas e localização utilizando o filtro de Kalman serão combinadas, implementando uma solução completa e integrando no ambiente de simulação V-Rep.

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, params, 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,
4. % given the previous state estimate, control inputs, laser measurements and
5. % the map
7. C\_TR = diag([repmat(0.1^2, 1, size(S, 2)) repmat(0.1^2, 1, size(S, 2))]);
8. [z, R, ans] = extractLinesPolar(S(1,:), S(2,:), C\_TR, params);

11. %STARTRM
12. figure(2), cla, hold on;
14. %compute z\_prior
15. z\_prior=[];
16. % Now we need to loop through the map.
17. nM = size(M, 2); % number of map entries.
19. % Loop through all Map Entries
20. **for** i=1:nM
21. % Call measurement function **for** each set of map entries.
22. [h, ~]= measurementFunction(x, M(:,i));
23. z\_prior = [z\_prior; h(1) h(2)];
24. end
25. % Just rearrange data at the end **for** plotting data.
26. z\_prior = z\_prior';
28. plot(z(1,:), z(2,:),'bo');
29. plot(z\_prior(1,:), z\_prior(2,:),'rx');
30. xlabel('angle [rad]'); ylabel('distance [m]')
31. legend('measurement','prior')
32. drawnow
34. % estimate robot pose
35. % Use the function from the previous step here.
36. [x\_posterori, P\_posterori] = filterStep(x, P, u, z, R, M, k, g, l);
38. %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