Universidade Estadual de Campinas
UNICAMP

IA368N

Atividade 4

Line-based Extended Kalman Filter for robot localization

Vinicius Rodrigues Sanches – RA: 208976

Introdução Teórica

O filtro de Kalman Estendido possibilita a fusão de sensores Gaussianos através de uma sequência de ações de predição (*prediction update*) e medição (*measurement update*). As ações de predição são meramente uma aplicação do modelo Gaussiano ao movimento previsto. Já o *measurement update* composto por algumas fases como *observation step*, *measurement prediction, matching step* e *estimation step*. Ele apresenta uma vantagem sobre o filtro de Kalman tradicional pois consegue fazer a linearização de sistemas não lineares.

Objetivo

Implementarei (complementarmente) um controle de localização baseado em EKF utilizando um mapa previamente conhecido e posição inicial conhecida. Vamos, com esse cálculo, atualizar a posição de um robô fantasma no simulador que nos indica visualmente onde o algoritmo está estimando a posição do robô real no mapa. O mapa no nosso caso será representado por um mapa de retas expressas em um sistema de coordenadas polares.

Predição do Estado

Na predição do estado utilizarei a função de transferência baseada na cinemática descritiva do robô, assim como seus principais Jacobianos. Para um robô diferencial com estados $\boldsymbol{x}_{t-1} = [x_{t-1}, y_{t-1}, \vartheta_{t-1}]^T$ e com ações de estado de suas rodas $\boldsymbol{u}_t = [\Delta s_l , \Delta s_r]^T$ o modelo cinemático para obter a predição de estado à priori segue abaixo:

$$\hat{\boldsymbol{x}}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) = \boldsymbol{x}_{t-1} + \begin{bmatrix} (\Delta s_l + \Delta s_r)/2.\cos(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2l) \\ (\Delta s_l + \Delta s_r)/2.\sin(\theta_{t-1} + (\Delta s_r - \Delta s_l)/2l) \\ (\Delta s_l - \Delta s_r)/l \end{bmatrix}$$

$$\boldsymbol{Q} = \begin{bmatrix} k|\Delta s_l| & 0 \\ 0 & k|\Delta s_l| \end{bmatrix}$$

Portanto temos a estimativa a priori da matriz de covariância como abaixo:

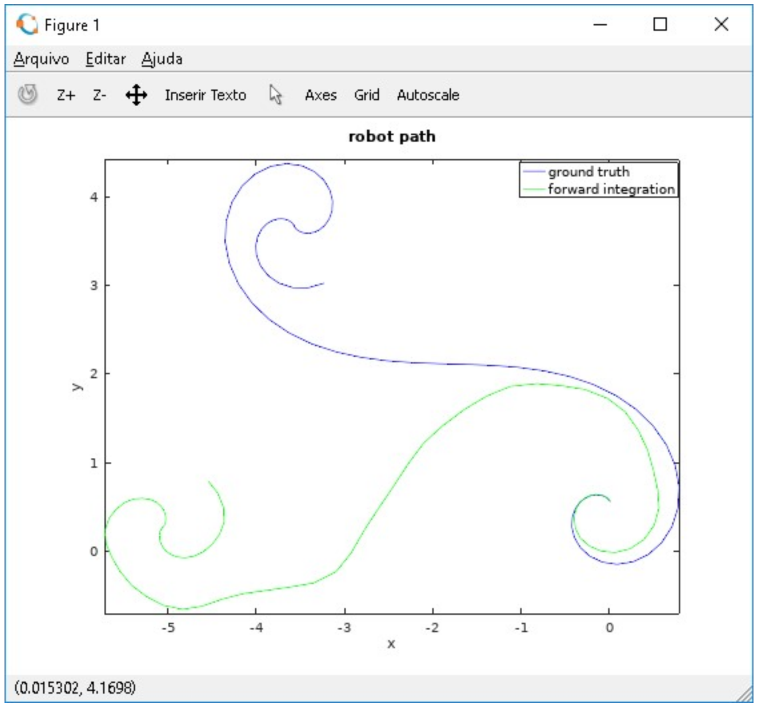
$$\widehat{\boldsymbol{P}}_t = \boldsymbol{F}_x . \boldsymbol{P}_{t-1} . \boldsymbol{F}_x{}^T + \boldsymbol{F}_u . \boldsymbol{Q}_t . \boldsymbol{F}_u{}^T$$

Onde F_x e F_u são os Jacobianos de modelo cinemático. Note que a constante k é usada para levar em conta efeitos não determinísticos.

Para conseguir tais Jacobianos tive de alterar o arquivo da função de transição, realizar a construção dos Jacobianos necessários e retorná-los. O código fica como segue abaixo:

```
function [f, F_x, F_u] = transitionFunction(x,u, l)
% [f, F_x, F_u] = transitionFunction(x,u,l) predicts the state x at time t
given
% the state at time t-1 and the input u at time t. F_x denotes the Jacobian
% of the state transition function with respect to the state evaluated at
% the state and input provided. F_u denotes the Jacobian of the state
% transition function with respect to the input evaluated at the state and
% input provided.
% State and input are defined according to "Introduction to Autonomous Mobile
Robots", pp. 337

%STARTRM
%pkg load symbolic
%
%syms Delta_r Delta_l th X Y b Delta_S Delta_th
b = l/2;
Delta_r = u(2);
Delta_l = u(1);
Delta_S = ((Delta_r + Delta_l)/2);
Delta_th= ((Delta_r - Delta_l)/l);

X = x(1);
Y = x(2);
th = x(3);

f = [X Y th]'+[ (((Delta_r + Delta_l)/2)*cos(th+(((Delta_r -
Delta_l)/(2*b))/2))); ...
                (((Delta_r + Delta_l)/2)*sin(th+(((Delta_r -
Delta_l)/(2*b))/2))); ...
                ((Delta_r - Delta_l)/(2*b))];




F_x = [ 1 0 -(Delta_S*sin(th+(Delta_th/2))); ...
        0 1 (Delta_S*cos(th+(Delta_th/2))); ...
        0 0 1];


V1 = cos(th+(Delta_th/2))/2 + (Delta_S/(2*l))*sin(th+(Delta_th/2));
V2 = cos(th+(Delta_th/2))/2 - (Delta_S/(2*l))*sin(th+(Delta_th/2));
V3 = sin(th+(Delta_th/2))/2 - (Delta_S/(2*l))*cos(th+(Delta_th/2));
V4 = sin(th+(Delta_th/2))/2 + (Delta_S/(2*l))*cos(th+(Delta_th/2));
F_u = [V1 V2; V3 V4; (-1/(l)) (1/(l))] ;

%ENDRM
```

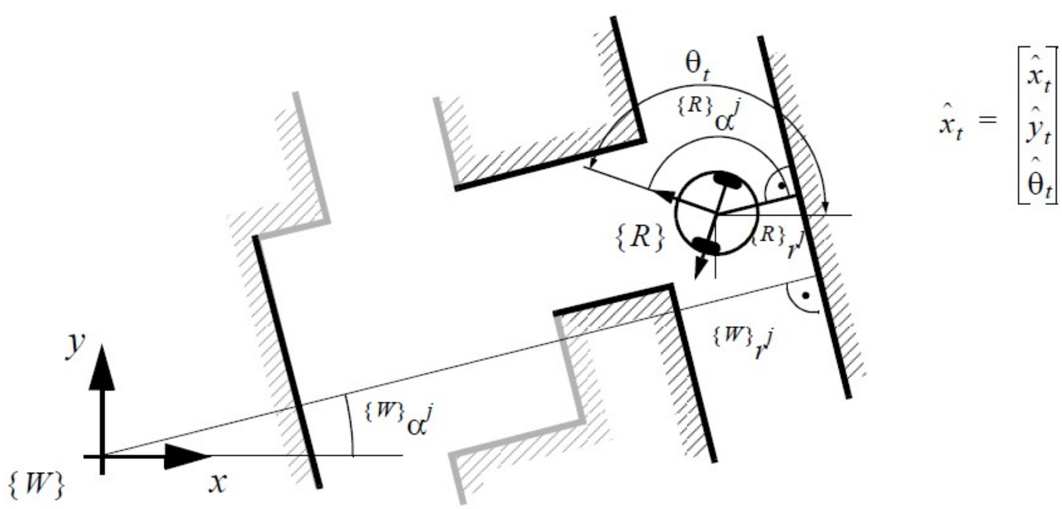
Lembrando que o Jacobiano F_u ficou com os sinais trocados em relação aos modelos tradicionais por conta da sequencia do vetor de ações u que coloca primeiro a velocidade da roda esquerda ao invés da direita, como é de costume na literatura. A função acima foi validada com sucesso pela função validateTransitionFunction e forma a figura baixo:



Atualização de Estado

Função de medição

Como na Atividade 3, as linhas são representadas pelo sistema polar de ângulo-distância. Essa representação é a mesma tanto para o nosso mapa M quanto para as linhas extraídas do nosso sistema de percepção. A única diferença é que o mapa nos dará os dados das linhas em relação ao frame global e o sensor em relação ao frame do próprio robô. Temos, portanto, que transformar um sistema no outro para podermos comparar os dados.



$$\hat{z}_t^j = \begin{bmatrix} \hat{\alpha}_t^j \\ \hat{r}_t^j \end{bmatrix} = h^j(\hat{x}_t, m^j) = \begin{bmatrix} {}^{\{W\}}\alpha_t^j - \hat{\theta}_t \\ {}^{\{W\}}r_t^j - (\hat{x}_t \cos({}^{\{W\}}\alpha_t^j) + \hat{y}_t \sin({}^{\{W\}}\alpha_t^j)) \end{bmatrix}$$

$$H^j = \begin{bmatrix} \frac{\partial \alpha_t^j}{\partial \hat{x}} & \frac{\partial \alpha_t^j}{\partial \hat{y}} & \frac{\partial \alpha_t^j}{\partial \hat{\theta}} \\ \frac{\partial r_t^j}{\partial \hat{x}} & \frac{\partial r_t^j}{\partial \hat{y}} & \frac{\partial r_t^j}{\partial \hat{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 \\ -\cos({}^{\{W\}}\alpha_t^j) & -\sin({}^{\{W\}}\alpha_t^j) & 0 \end{bmatrix}$$

Foi implementada a função *measurementFunction* com sucesso. Ela realiza todos os cálculos e transformações acima, retornando a as linhas previstas no mapa em relação à posição prevista anteriormente pela função de transição. Segue a implementação abaixo:

```
function [h, H_x] = measurementFunction(x, m)
% [h, H_x] = measurementFunction(x, m) returns the predicted measurement
% given a state x and a single map entry m. H_x denotes the Jacobian of the
% measurement function with respect to the state evaluated at the state
% provided.
% Map entry and state are defined according to "Introduction to Autonomous
Mobile Robots" pp. 337

%STARTRM
h = [m(1)-x(3); m(2)-(x(1)*cos(m(1))+x(2)*sin(m(1)))];

H_x = [0 0 -1; -cos(m(1)) -sin(m(1)) 0];

%ENDRM

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

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

Associação de Medições

Nesse passo tem que se criar uma associação entre o que está sendo observado e o que foi previsto de acordo com o mapa. Para tal fim se calcula a *innovation* entre as medições previstas e as observadas para então aplicar a distância de *Mahalanobis*, como seguem as expressões abaixo:

$$\boldsymbol{v}_t^{ij} = \boldsymbol{z}_t^j - \hat{\boldsymbol{z}}_t^i$$

E uma nova covariância da inovação (*innovation covariance*):

$$\boldsymbol{\Sigma}_{IN_t}^{ij} = \widehat{\boldsymbol{H}}_t^i . \widehat{\boldsymbol{P}}_t . \widehat{\boldsymbol{H}}_t^{iT} + \boldsymbol{R}_t^j$$

A distância de *Mahalanobis* é dada por:

$$\mathrm{d}_t^{ij} = \boldsymbol{v}_t^{ijT} . \left(\boldsymbol{\Sigma}_{IN_t}^{ij}\right)^{-1} . \boldsymbol{v}_t^{ij}$$

Na função *associateMeasurements* podemos observar uma busca entre todos os pontos do mapa previstos e todos os pontos observados na realidade. A função recebe um parâmetro g que faz uma limitação dos valores aceitos associados. Essa função foi testada com sucesso pela função *validateAssociations*.

Atualização da Estimativa

Nesse ponto foi implementada parte da função do Filtro de Kalman Estendido que une todas as etapas previamente descritas (*filterStep.m*), atualizando as posições do robô e sua covariância associada de acordo com uma estimativa feita pelo filtro.

$$x_t = \hat{x}_t + K_t v_t ,$$

$$P_t = \hat{P}_t - K_t \cdot \Sigma_{IN_t} \cdot K_t^T ,$$

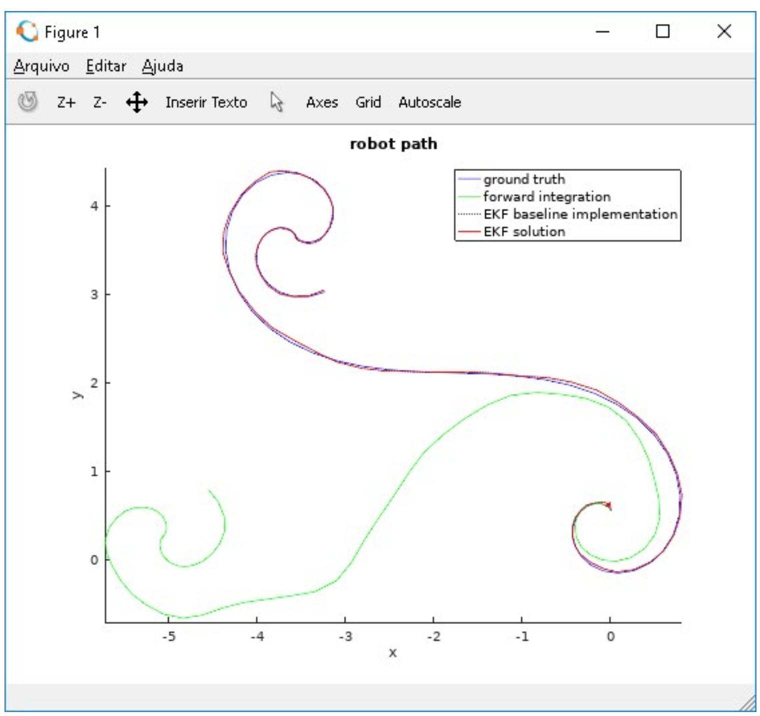
Onde K é o ganho de Kalman:

$$K_t = \hat{P}_t \cdot H_t^T \cdot (\Sigma_{IN_t})^{-1}$$

Segue a implementação e resultado de sua avaliação feita pela função *validateFilter* da função *filterStep*.

```
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


Delta_l = u(1);
Delta_r = u(2);
% propagate the state (p. 337) , here kr=kl=k
Q = [k*abs(Delta_r) 0; 0 k*abs(Delta_l)];

[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);

y = reshape(v, [], 1);
H = reshape(permute(H, [1,3,2]), [], 3);
R = blockDiagonal(R);

% update state estimates (pp. 335)
S = (H*P_priori*(H'))+R;
K = P_priori*(H')*(inv(S));

x_posteriori = x_priori + K * y;
P_posteriori = (eye(size(P_priori)) - K*H) * P_priori;
```

Resultado:



Podemos verificar nessa figura que o resultado da linha da solução do EKF passa muito próxima a linha real e da baseline EKF, sendo visivelmente muito melhor que a forward integration pura.

Experimento no V-REP

Nesse passo foi complementada a implementação de uma função *incrementalLocalization* para um experimento utilizando o V-REP. Essa função nos permite realizar a execução do EKF em ambiente simulado utilizando o arquivo de testes de simulação *vrepSimulation.m*. Essa implementação vai reunir todos os conceitos abordados até agora desde as atividades 2 e 3.

```
function [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S, M,
params, k, g, l)
% [x_posterori, P_posterori] = incrementalLocalization(x, P, u, S, R, M,
% 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=[];
nMapEntries = size(M,2);
for i=1:nMapEntries
    [z_i, H]= measurementFunction(x, M(:,i));
    z_prior = [z_prior; z_i(1) z_i(2)];
end
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
[x_posterori, P_posterori] = filterStep(x, P, u, z, R, M, k, g, l);#; %hint:
you just coded this function

%ENDRM
```

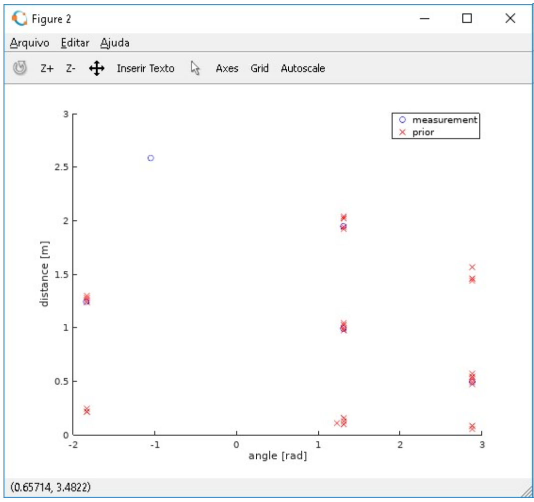
O resultado dessa função é utilizado para alterar a posição e covariância associada do robô fantasma da simulação no arquivo *vrepSimulation.m*. com o comando abaixo:

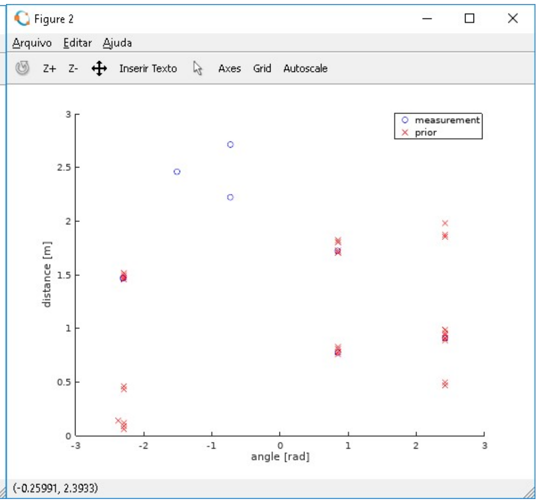
```
    [x, P] = incrementalLocalization(x, P, u, [theta'; rho'], M, params, k, g,
l);

    % plot pose estimate in vrep
    Pioneer_p3dx_setGhostPose(connection, x(1), x(2), x(3));
```

Houveram muitos passos de difícil depuração nesse ponto da Atividade 4 devido as diferenças de comportamento dos filtros de imagens que geram o mapa M entre o Octave e o Matlab.

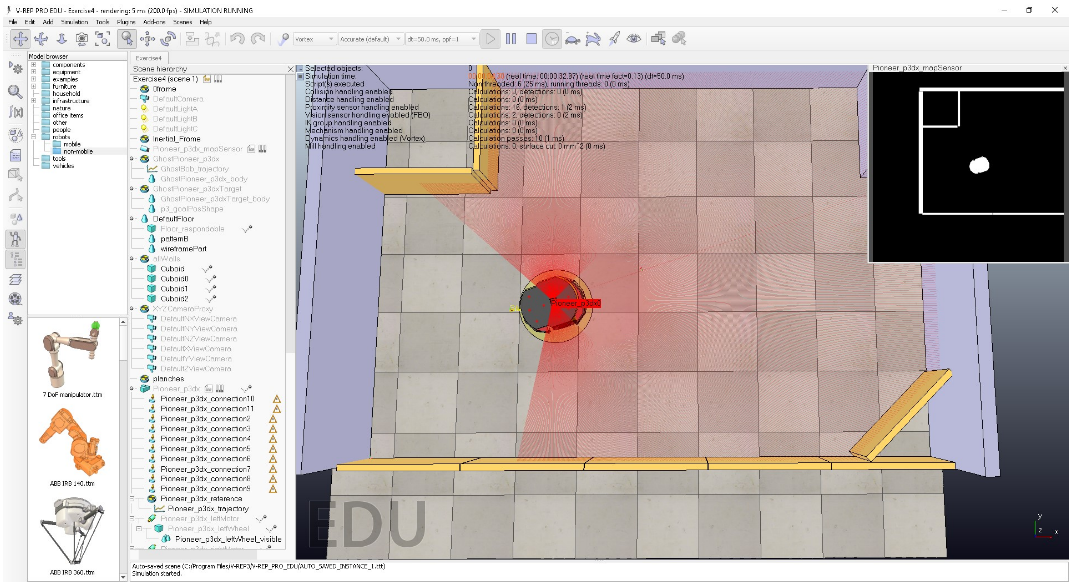
Atualmente estou usando o Octave e tive de alterar o filtro de imagens para o Canny no *generateMap.m*. Outra diferença básica foi na alteração das velocidades das rodas para que o Octave não parasse de funcionar(Crash).

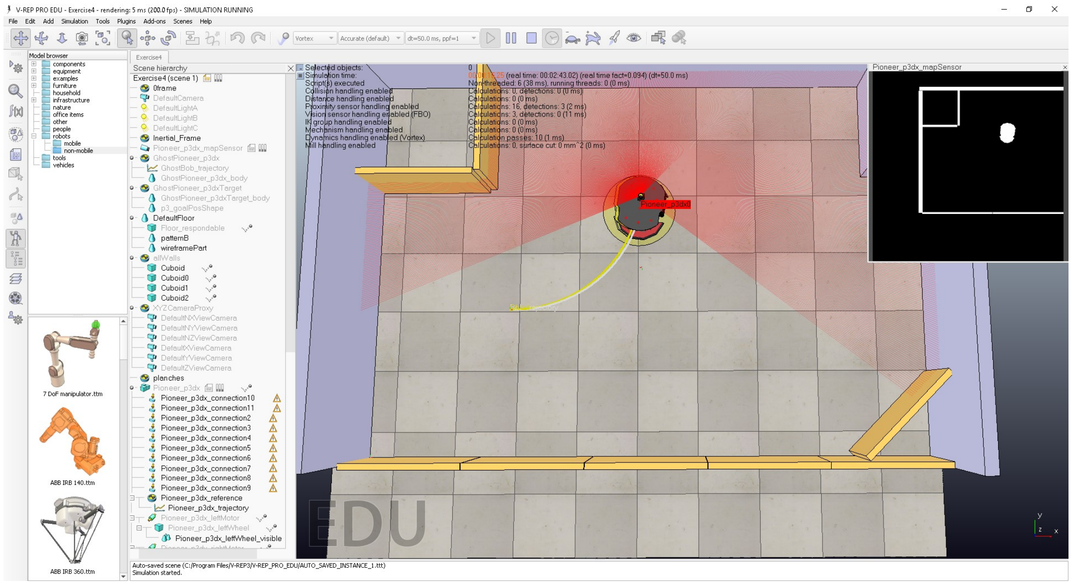
Depois de muito pesquisar uma solução de implementação foi conseguido o resultado esperado.

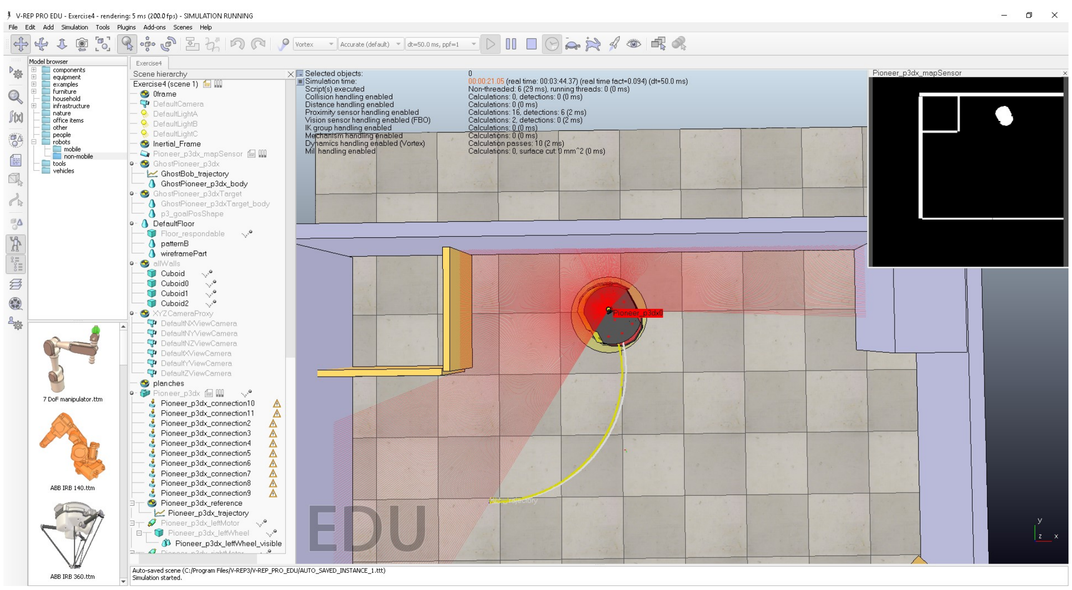




Notamos aqui que existem muitos pontos de Z a priori devido ao numero muito grande de linhas extraídas da imagem filtrada. Vemos também as linhas que foram medidas durante o trajeto do robô que não tem correspondência com nenhuma extraída do mapa pois o mapa gerado para a experiência não possui todas as linhas possíveis do mapa medido em tempo real. O filtro retorna tanto as linhas internas do mapa quando as externas e talvez por isso a localização fique prejudicada na exatidão. Mas mesmo assim podemos verificar uma boa precisão depois de 400 ciclos.







Conclusões

O EKF utilizando um mapa baseado em linhas representadas em coordenadas polares se mostrou eficaz para a localização e de razoável facilidade de implementação.

Source code no GitHub

https://github.com/viniciusrsanches/IA368N

Bibliografia

Siegwart, Roland.

Introduction to autonomous mobile robots. - 2nd ed. / Roland Siegwart, Illah R. Nourbakhsh, and Davide

Scaramuzza.

p. cm. - (Intelligent robotics and autonomous agents series)

Slides de aula IA368N – Professor Eric Rohmer