Universidade Estadual de Campinas
UNICAMP

IA368N

Atividade 2
Kinematics and control of a differential drive vehicle

Vinicius Rodrigues Sanches – RA: 208976

Introdução Teórica

Robôs diferenciais são muito utilizados em pesquisas ou até mesmo em ambientes fabris por se tratarem de um tipo muito particular de plataforma de locomoção que facilita muito a construção do modelo matemático e por fim de todos os cálculos envolvidos.

Objetivo

O objetivo nessa atividade é construir um controlador de robôs diferenciais em loop fechado. Para isso, três sub tarefas tem de ser resolvidas com sucesso.

Introdução aos tópicos subsequentes.

1) Dadas as velocidades direta e angular desejadas, deve-se calcular as velocidades das rodas correspondentes para fazer o robô dirigir em conformidade.

2) Com o resultado da tarefa anterior, você deve desenvolver uma operação de tele operação do robô através do teclado.

3) Desenvolva um controlador que computa comandos de velocidade para dirigir o robô para uma posição alvo especificada. As soluções serão desenvolvidas e implementadas em Matlab / Octave e testadas dentro de um ambiente de simulação.

Controle de Avanço

Para um robô de direção diferencial, o modelo cinemático é descrito pelas seguintes

equações:

$$v = \frac{r\varphi_r}{2} + \frac{r\varphi_l}{2}$$

$$\omega = \frac{r\varphi_r}{2l} - \frac{r\varphi_l}{2l}$$

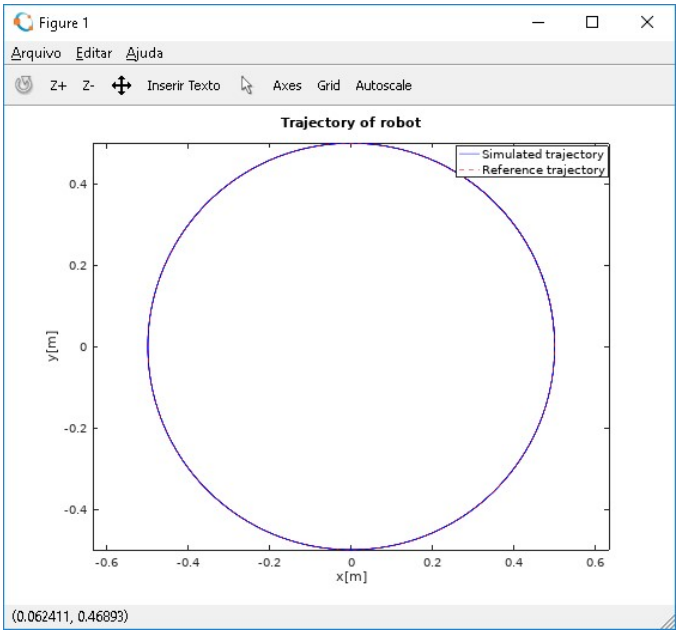
onde (v, ω) representam a velocidade de deslocamento e velocidade angular da plataforma do robô, respectivamente, e (ϕr, ˙ ϕl) a velocidade de rotação das rodas direita e esquerda. O raio da roda é dado por r e l denota a metade da distância inter-roda.

Construirei um controlador que, com esses parâmetros, o robô efetue uma curva de 0.5m de raio. Para isso vamos editar o arquivo **code/commom/vrep/calculateWheelSpeeds.m** e validar com o arquivo **/code/common/vrep/testCircleDrive.m**.

Dadas as equações anteriores, se recebermos as entradas vu e ω temos somente que resolver o sistema de equações. Criamos um vetor velocidade v contendo as velocidades vu (velocidade de deslocamento) e ω (velocidade angular de giro) e colocamos todos os parâmetros de configuração da geometria e modelo cinemático do robô numa matriz M. Logo ficamos com um sistema simples de 2 variáveis e 2 equações que podemos facilmente resolver como segue o código fonte:

```
function [ LeftWheelVelocity, RightWheelVelocity ] = calculateWheelSpeeds( vu,
omega, parameters )
%CALCULATEWHEELSPEEDS This function computes the motor velocities for a
differential driven robot

wheelRadius = parameters.wheelRadius;
halfWheelbase = parameters.interWheelDistance/2;

M = [parameters.wheelRadius/2 parameters.wheelRadius/2;
parameters.wheelRadius/(parameters.interWheelDistance) -
parameters.wheelRadius/(parameters.interWheelDistance)];
v = [vu; omega];
wheels_velocity = M \ v;
LeftWheelVelocity = wheels_velocity(2);
RightWheelVelocity = wheels_velocity(1);

end
```

Como podemos observar, resolvemos o sistema de equações usando a linha `wheels_velocity = M \ v;` calculando a inversa de M e multiplicando por v, resultando num vetor com as velocidades das rodas. Essas velocidades podem ser passadas para o simulador dizendo o quão rápido as rodas devem girar para termos a velocidade vu e a velocidade angular ω.



Como pudemos observar, o resultado ficou preciso. O circulo está passando exatamente pela referência.

Tele Operação

Nesse exercício foi construído um simples controle manual com 5 comandos básicos:

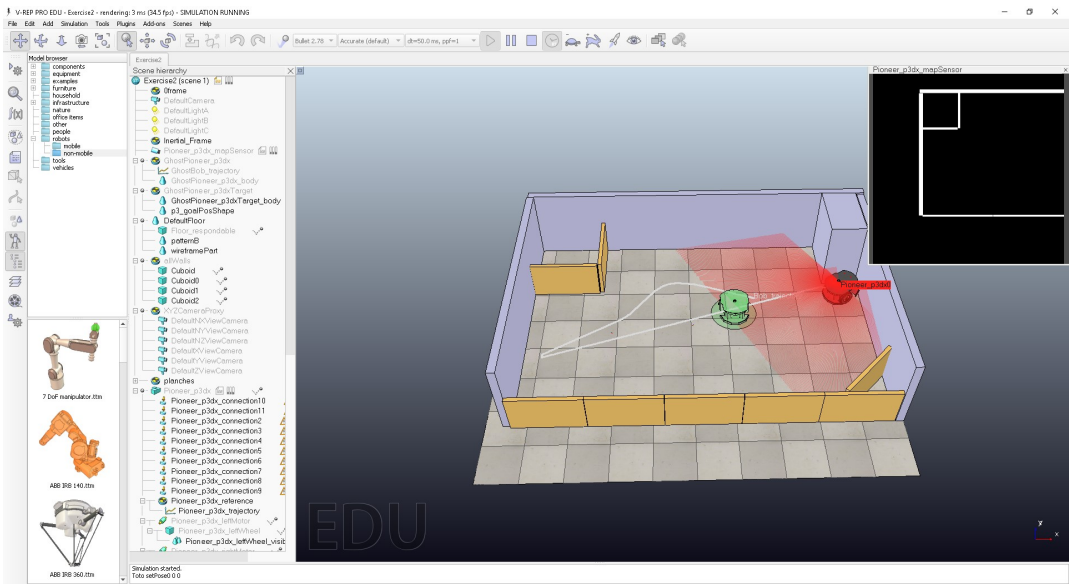
1. Aumenta velocidade de avanço
2. Aumenta velocidade de recuo
3. Aumenta velocidade do sentido horário
4. Aumenta velocidade do sentido anti-horário
5. Parada imediata

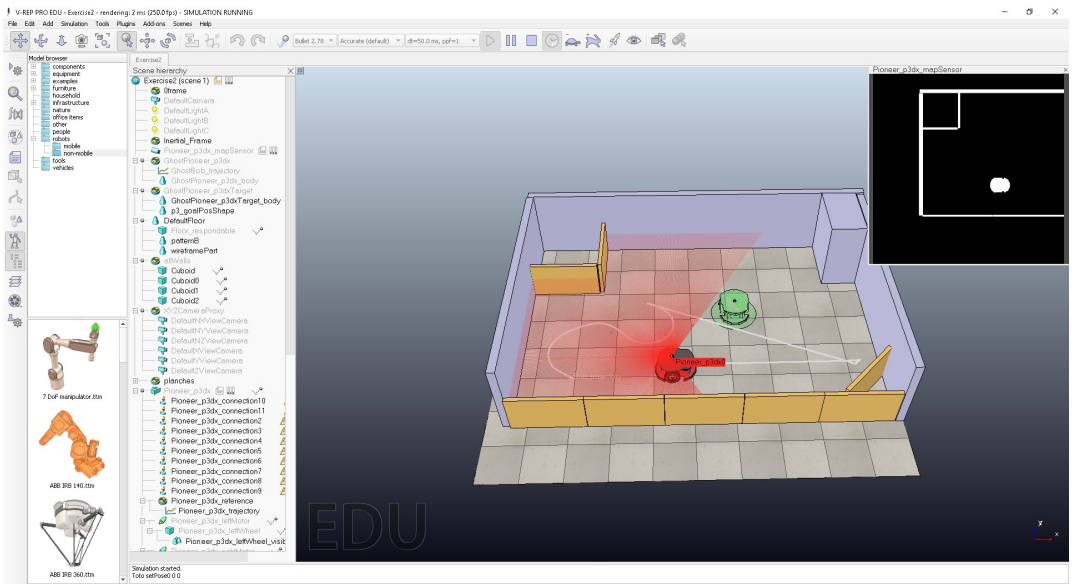
O código abaixo utiliza somente um simples controle de teclas para alterar os parâmetros que serão passados para a função que construímos anteriormente, pela qual vai gerar os valores de velocidades das rodas que serão atualizados no robô do simulador. Vemos abaixo o trecho de código resultante:

```
%% teleoperation program goes here
%% CONTROL LOOP.
EndCond = 0;
vu = 0;
omega = 0;
while (~EndCond)
    %% CONTROL STEP.
    % Get pose and goalPose from vrep
    [x, y, theta] = Pioneer_p3dx_getPose(connection);
    [xg, yg, thetag] = Pioneer_p3dx_getTargetGhostPose(connection);

    % run control step
```

```
    %[ vu, omega ] = calculateControlOutput([x, y, theta], [xg, yg, thetag],
parameters);

    key = kbhit(1);
    if key == 'w' && vu < 0.25
      vu += 0.05;
    elseif key == 's' && vu > -0.25
      vu -= 0.05;
    elseif key == 'a' && omega < 0.75
%       if vu >= 0
         omega += 0.05;
%       else
%         omega -= 0.05;
%       end
    elseif key == 'd' && omega > -0.75
%       if vu >= 0
         omega -= 0.05;
%       else
%         omega += 0.05;
%       end
    elseif isempty(key)
%       if vu > 0
%         vu -= 0.001;
%       elseif vu < 0
%         vu += 0.001;
%       end
      if omega > 0
        omega -= 0.0025;
      elseif omega < 0
        omega += 0.0025;
      end
    end

    %Avoinding range operation overflow
    if abs(vu) < 0.01
      vu = 0;
    end
    if abs(omega) < 0.005
      omega = 0;
    end
    % Calculate wheel speeds
    [LeftWheelVelocity, RightWheelVelocity ] = calculateWheelSpeeds(vu, omega,
parameters);

    % End condition
    dtheta = abs(normalizeAngle(theta-thetag));

    rho = sqrt((xg-x)^2+(yg-y)^2);  % pythagoras theorem, sqrt(dx^2 + dy^2)
    EndCond = (rho < parameters.dist_threshold && dtheta <
parameters.angle_threshold);
    if key == ' '
      EndCond = 1;
    end
    % SET ROBOT WHEEL SPEEDS.
    Pioneer_p3dx_setWheelSpeeds(connection, LeftWheelVelocity,
RightWheelVelocity);
end
```

Vemos agora algumas figuras dos caminhos gerados utilizando os comandos manuais:

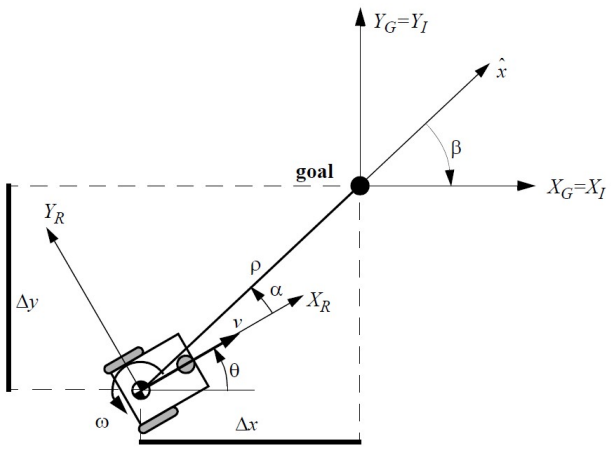




Como podemos observar, temos o controle total de todas as manobras possíveis que o robô pode executar.

Controle de loop fechado

Será feito agora um controlador proporcional que baseado na distância e na posição do robô em relação ao alvo decide, através de um sistema linear, quais os valores de velocidade linear e velocidade angular. O sistema leva em consideração os ângulos alpha, beta e theta assim como a distância do alvo p. O sistema deve ser suavizado por constantes K_p , K_{alpha} e K_{beta} como mostrado nas equações abaixo:



$$v = k_\rho \rho$$

$$\omega = k_\alpha \alpha + k_\beta \beta$$

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}\,.$$

$$\alpha = -\theta + \operatorname{atan2}(\Delta y, \Delta x)\,.$$

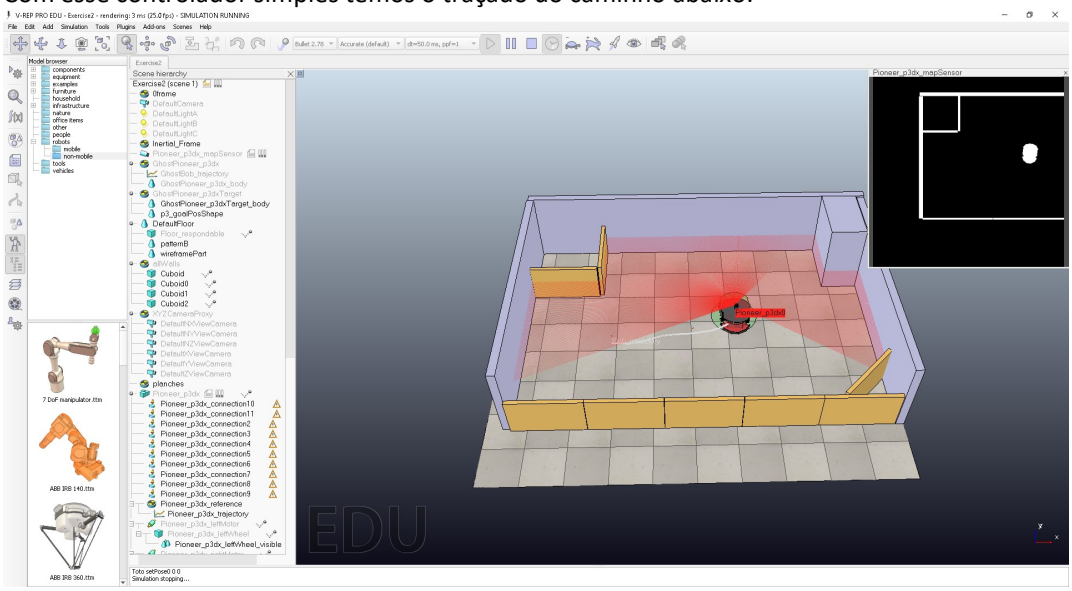
$$\beta = -\theta - \alpha\,.$$

Seguindo as restrições, temos uma função que calcula o valor dos controles em **code/commom/vrep/calculateControlOutput.m** que ficou como segue:

```
function [ vu, omega ] = calculateControlOutput( robotPose, goalPose,
parameters )
%CALCULATECONTROLOUTPUT This function computes the motor velocities for a
differential driven robot
%global count_aux;
%global ahead;
% current robot position and orientation
x = robotPose(1);
y = robotPose(2);
theta = robotPose(3);

% goal position and orientation
xg = goalPose(1);
yg = goalPose(2);
thetag = goalPose(3);

% compute control quantities
rho = sqrt((xg-x)^2+(yg-y)^2);  % pythagoras theorem, sqrt(dx^2 + dy^2)
lambda = atan2(yg-y, xg-x);     % angle of the vector pointing from the robot
to the goal in the inertial frame
```

```
alpha = lambda - theta;          % angle of the vector pointing from the robot
to the goal in the robot frame
alpha = normalizeAngle(alpha);

% the following paramerters should be used:
% Task 3:
% parameters.Kalpha, parameters.Kbeta, parameters.Krho: controller tuning
parameters
% Task 4:
% parameters.backwardAllowed: This boolean variable should switch the between
the two controllers
% parameters.useConstantSpeed: Turn on constant speed option
% parameters.constantSpeed: The speed used when constant speed option is on


vu = parameters.Krho* rho % [m/s]
omega = parameters.Kalpha*alpha + parameters.Kbeta *
(normalizeAngle(normalizeAngle(-lambda)+thetag));% [rad/s]
```

Com esse controlador simples temos o traçado do caminho abaixo:



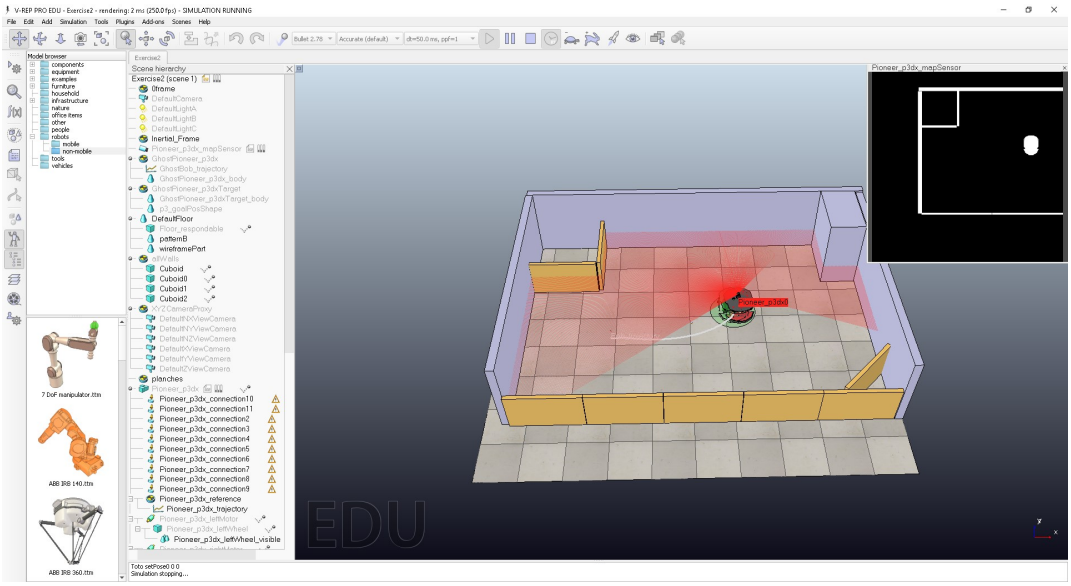
Pude observar que esse controlador, em algumas situações, tende a se tornar pouco eficiente quando está muito perto do alvo pois sua velocidade tende a zero e também por não conseguir andar de ré. Esses pontos serão corrigidos no próximo tópico.

Controle de Loop Fechado Melhorado

Vou agora alterar a mesma função para termos uma velocidade constante e quando o alvo estiver atrás do robô, o mesmo fará uma manobra (giro) com velocidade negativa (ré), para facilitar o caminho e a finalização no alvo. Simplesmente substituímos as linhas anteriores pelas abaixo e adicionamos uma condição de velocidade negativa quando o alvo estiver atrás do robô:

```
function [ vu, omega ] = calculateControlOutput( robotPose, goalPose,
parameters )
%CALCULATECONTROLOUTPUT This function computes the motor velocities for a
differential driven robot
%global count_aux;
%global ahead;
% current robot position and orientation
x = robotPose(1);
y = robotPose(2);
theta = robotPose(3);

% goal position and orientation
xg = goalPose(1);
yg = goalPose(2);
thetag = goalPose(3);

% compute control quantities
rho = sqrt((xg-x)^2+(yg-y)^2);  % pythagoras theorem, sqrt(dx^2 + dy^2)
lambda = atan2(yg-y, xg-x);     % angle of the vector pointing from the robot
to the goal in the inertial frame
alpha = lambda - theta;         % angle of the vector pointing from the robot
to the goal in the robot frame
alpha = normalizeAngle(alpha);

% the following paramerters should be used:
% Task 3:
% parameters.Kalpha, parameters.Kbeta, parameters.Krho: controller tuning
parameters
% Task 4:
% parameters.backwardAllowed: This boolean variable should switch the between
the two controllers
% parameters.useConstantSpeed: Turn on constant speed option
% parameters.constantSpeed: The speed used when constant speed option is on


vu = 0.3;%parameters.Krho* rho; % [m/s]
omega = parameters.Kalpha*alpha + parameters.Kbeta *
(normalizeAngle(normalizeAngle(-lambda)+thetag));% [rad/s]

if alpha <= -pi/2 && alpha > pi/2
 vu = -vu;
end
```

Com isso temos um caminho mais direto e sem muita variação de velocidade. Outro ponto é a capacidade de executar velocidade negativa quando satisfeita a última condição (alvo se encontra relativamente atrás do robô).



Conclusões

O sistema demonstrado se mostrou bem eficiente para as situações simples demonstradas aqui nesse experimento. Outro fator importante é que se faz muito simples de implementar, suavizando caminhos. Outro ponto observado é que esse controle deveria ser utilizado em conjunto com um path planner, o qual ofereceria os pontos passo a passo, pois esse sistema não prevê obstáculos.

Source code on 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