Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

IA903 – Introdução à Robótica Móvel 29 de setembro de 2024

Docente: Eric Rohmer

Discente:

– Gabriel Toffanetto França da Rocha – 289320

Exercise 2: Kinematics and control of a differential drive vehicle

---

Sumário

---

1 Cinemática inversa do robô diferencial

A cinemática direta do robô diferencial expressa como a velocidade linear (v) e a velocidade angular (ω) do robô se relacionam com a velocidades das suas rodas, φ_l e φ_r, que possuem raio r e estão à uma distância l do CG do robô. Tais relações são definidas em (1) e (2), sendo compactadas de forma matricial em (3).

$$v = \frac{r\varphi_r}{2} + \frac{r\varphi_l}{2} \tag{1}$$

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

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix} \begin{bmatrix} \varphi_r \\ \varphi_l \end{bmatrix} \tag{3}$$

Para que dada uma velocidade linear e angular desejas, o robô seja capaz de seguí-la, precisa-se saber qual a velocidade de cada roda deve desenvolver. Para isso, é utilizada a cinemática inversa, obtida ao isolar o vetor de velocidade das rodas em (3). Realizando tal procedimento, de acordo com (4), é obtida a relação da velocidade linear e angular que levam à velocidade das rodas do robô.

$$\begin{bmatrix} \varphi_r \\ \varphi_l \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix}^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix} = \frac{\text{adj}\left(\begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix}\right)}{\det\left(\begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix}\right)} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{l}{r} \\ \frac{1}{r} & -\frac{l}{r} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4}$$

Foi realizada a validação por meio de simulação do modelo, obtendo a Figura 1, que demonstra que o modelo conseguiu desempenhar a trajetória desejada.

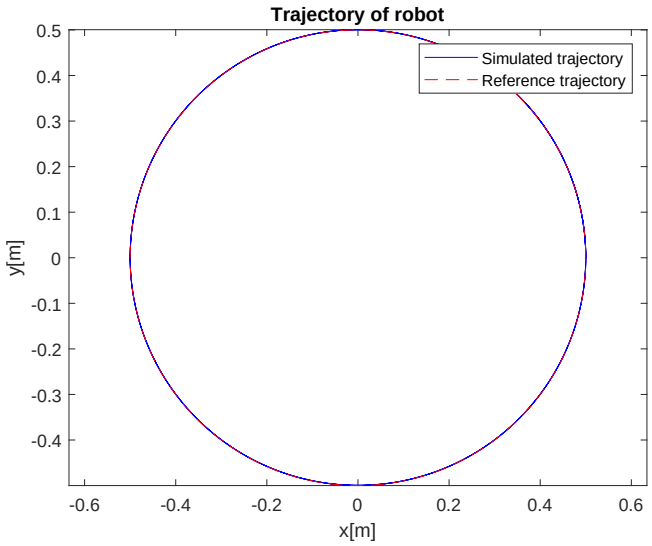

Figura 1: Validação da cinemática inversa do robô.

2 Teleoperação do robô diferencial

O algoritmo de teleoperação foi implementado de acordo com o Algoritmo 1.

Algoritmo 1 Algoritmo de teleoperação.

```
1: v ← 0;
2: ω ← 0;
3: K ← ReadKeyboardKey(;)
4: while K ≠ q do
5:     switch K do
6:         case w:
7:             v ← v + 0.1;
8:         case s:
9:             v ← v − 0.1;
10:        case a:
11:            ω ← ω + 0.1;
12:        case d:
13:            ω ← ω − 0.1;
14:        case SPACE:
15:            v ← 0;
16:            ω ← 0;
17:    [φ_l, φ_r] ← calculateWheelSpeeds(v, w, parameters);
18:    setWheelSpeeds(φ_l, φ_l)
19:    K ← ReadKeyboardKey(;)
20: end while
```

3 Controle em malha fechada

Aplicado o controle em malha fechada, dado por (5), obtêm-se o posicionamento do robô nas posições desejadas, como mostrado nas Figuras 2, 3 e 4.

$$\begin{aligned} \rho &= \sqrt{(x_g - x)^2 + (y_g - y)^2} \\ \alpha &= \mod\ [\text{atan2}\,(y_g - y, x_g - x) - \theta + \pi,\ 2\pi] - \pi \\ v &= k_\rho \rho \\ \omega &= k_\alpha \alpha \end{aligned} \tag{5}$$

Sendo $k_\rho = 0,5$ e $k_\alpha = 1.5$, respeitando as condições para estabilidade do sistema.

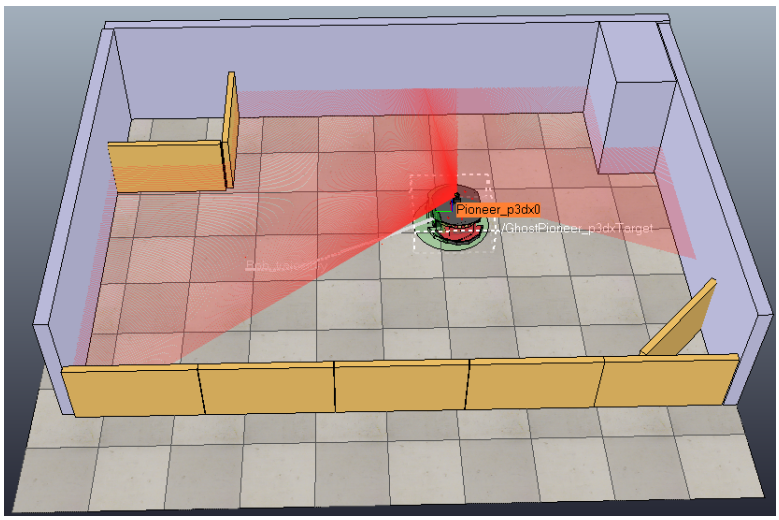

Figura 2: Trajetória do robô para a posição $(1,6; 0,6)$ orientado à 90°.

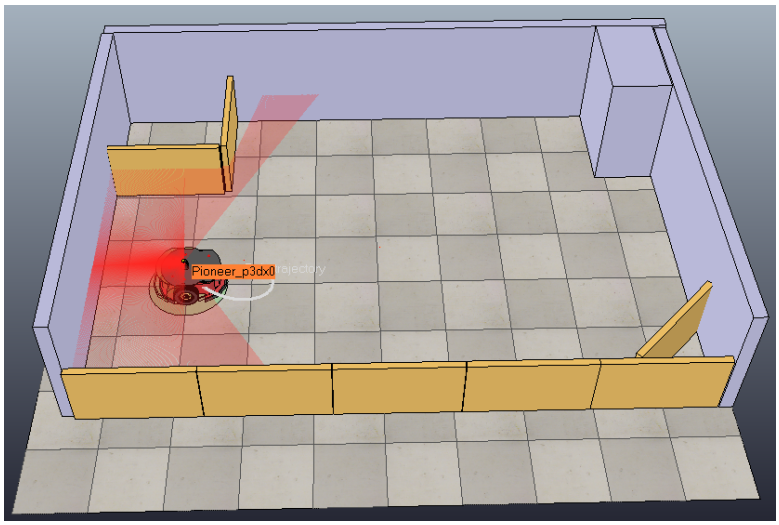

Figura 3: Trajetória do robô para a posição $(-0,5; 0)$ orientado à 180°.

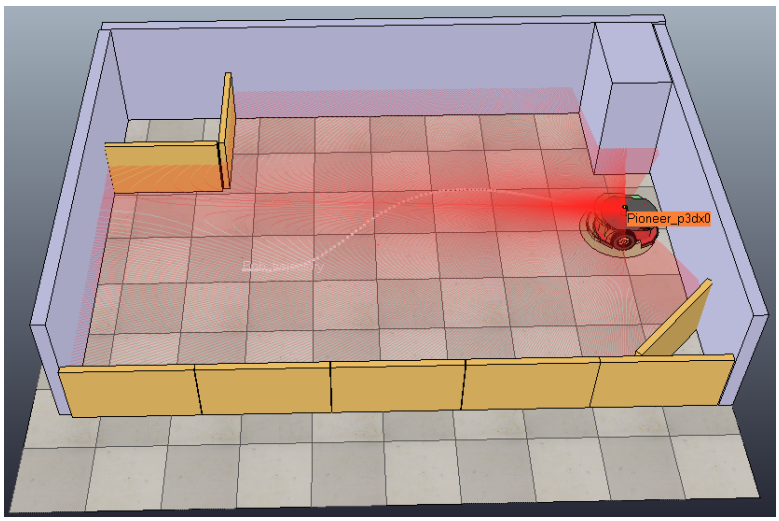

Figura 4: Trajetória do robô para a posição $(3; 0,5)$ orientado à 180°.

4 Controle melhorado

A implantação de um controle melhorado, que possibilite o robô se mover para trás e conduzir em velocidade linear constante foi feita em duas partes: (i) Verificando se o robô deve seguir para frente ou para trás em direção ao destino; (ii) Obtendo a velocidade linear e angular para que o robô navegue em velocidade constante.

O Algoritmo 2 verifica se o alvo se encontra à frente ou à trás do robô, por meio do ângulo de erro α entre a orientação do robô e a o ângulo do ponto de destino no sistema inercial. Com isso, caso $|\alpha| \leq \pi/2$, o mesmo deve se guiar para a frente, enquanto se $\alpha > \pi/2$, o mesmo deve se mover para trás. Para o segundo caso, a velocidade linear deve receber um sinal negativo, e o ângulo α ser acrecido de π, uma vez que o referencial de direção do robô passa à ser $-x$.

Algoritmo 2 Algoritmo de direção.

```
1: s_v ← 1;
2: if backwardAllowed then
3:     if |α| > π/2 then
4:         s_v ← −1;
5:         α ← NORMALIZEANGLE(α + π);
6:     end if
7: end if
```

Algoritmo 3 Algoritmo de velocidade.

```
1: s_v ← 1;
2: if useConstantSpeed then
3:     if ρ > dist_threshold ·2 then
4:         ω ← ω · (constantSpeed/v);
5:         v ← constantSpeed ·s_v;
6:     else
7:         ω ← ω_i;
8:         v ← v_i · s_v;
9:     end if
10: end if
```

A constante constantSpeed/v calcula o quanto a velocidade linear do robô dada pelo controle foi alterada pelo uso da velocidade constante, fazendo a mesma compensação na velocidade angular, com intuito de manter a razão v/ω constante. Onde v_i e ω_i são as velocidades lineares e angulares calculadas de acordo com o controle de (5). Foi necessário para um controle mais suave desativar a velocidade constante quando o robô está bem próximo do alvo, evitando a ocorrência de grandes velocidades angulares que geram *overshoot*.

Como resultado, observou-se uma trajetória mais suave até o alvo, mas porém, com um tempo de missão maior, como mostrado nas Figuras 5, 6 e 7.

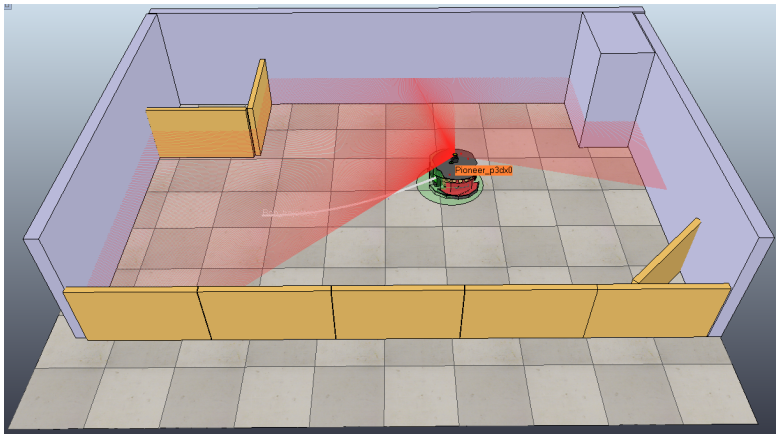

Figura 5: Trajetória do robô para a posição $(1,6; 0,6)$ orientado à 90°.

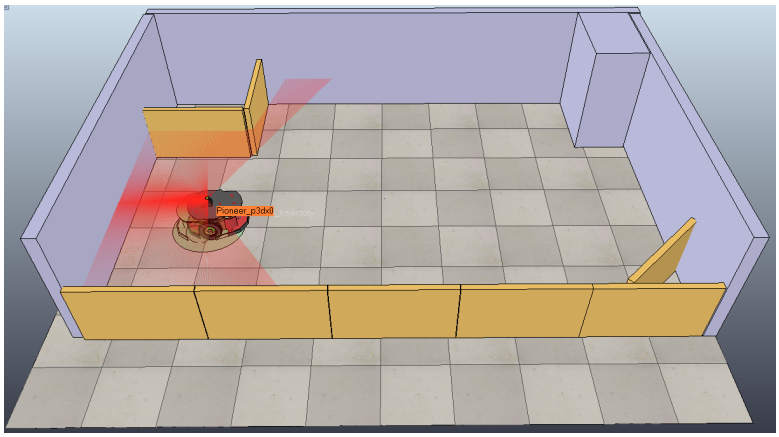

Figura 6: Trajetória do robô para a posição $(-0,5; 0)$ orientado à 180°.

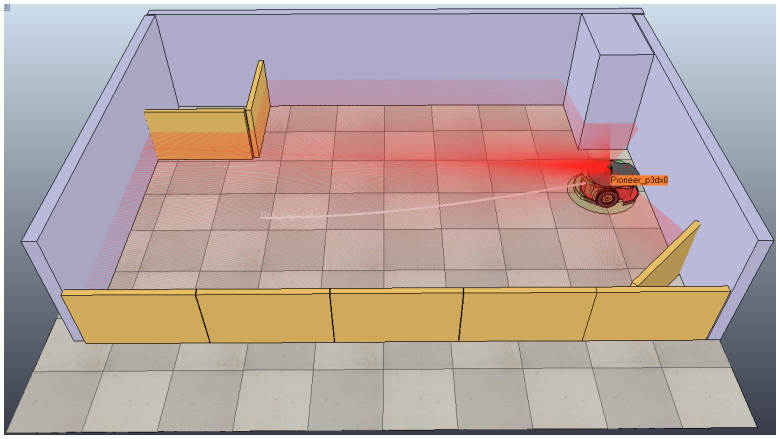

Figura 7: Trajetória do robô para a posição $(3; 0,5)$ orientado à 180°.

Apêndices

Código cinemática inversa

```
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;

LeftWheelVelocity = (1/wheelRadius)*vu - (halfWheelbase/wheelRadius)*omega;
RightWheelVelocity = (1/wheelRadius)*vu + (halfWheelbase/wheelRadius)*omega;

end
```

Código teleoperação

```
%% V-REP Simulation Exercise 2-3: Teleoperation
% Tests the algorithm within a V-Rep simulation.

% In order to run the simulation:
%   - Start V-Rep
%   - Load the scene /scene/Exercise2.ttt
%   - Hit the run button
%   - Start this script


clear;
close all;
%% Parameters setup

%% define that we will use the real P3DX and/or the simulated one
global realRobot ;
realRobot = 0; %realRobot=0 for simulation and realRobot=1 to use the real
    robot

global laserStr;
laserStr = '/perception/laser/';

global poseStr;
poseStr = '/motion/pose';

global vel2Str;
vel2Str = '/motion/vel2';

global stopStr;
stopStr = '/motion/stop';
```

```
30
31 global parameters;
32
33 %% initialization
34 if realRobot==1
35     % http_init will add the necessary paths
36     http_init;
37
38     % Declaration of variables
39     connection = 'http://10.1.3.130:4950';
40     %connection = 'http://143.106.11.166:4950';
41
42     parameters.wheelDiameter = .195;
43     parameters.wheelRadius = parameters.wheelDiameter/2.0;
44     parameters.interWheelDistance = .381/2;
45
46     %1% Pioneer_p3dx_setTargetGhostPose(connection, -2, 0, 0);
47
48     %% Set the initial pose of the robot ( TO CHANGE INTO A FUNCTION
    SET_POSE)
49     %1%http_put([connection '/motion/pose'], struct ('x',0,'y',-2 *1000,'th
    ',90)) ;
50     %http_put([connection '/motion/pose'], struct ('x',0,'y',0,'th',0)) ;
51     % parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
    connection);
52 else
53     %% Initialize connection with V-Rep
54     connection = simulation_setup();
55     connection = simulation_openConnection(connection, 0);
56     simulation_start(connection);
57
58     %% Get static data from V-Rep
59     Pioneer_p3dx_init(connection);
60     parameters.wheelDiameter = Pioneer_p3dx_getWheelDiameter(connection);
61     parameters.wheelRadius = parameters.wheelDiameter/2.0;
62     parameters.interWheelDistance = Pioneer_p3dx_getInterWheelDistance(
    connection);
63     parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
    connection);
64     Pioneer_p3dx_setTargetGhostVisible(connection, 1);
65
66 end
67 %initialization ends here
68
69 Pioneer_p3dx_setPose(connection, 0,0,0);
70
71 % define target position
72 %Pioneer_p3dx_setTargetGhostPose(connection, 0, -1, deg2rad(90));
73 %1%Pioneer_p3dx_setTargetGhostPose(connection, -1, 0, 0);
```

```
74 Pioneer_p3dx_setTargetGhostPose(connection, 1.6, 0.6, deg2rad(90));
75
76
77 % controller parameters
78 parameters.Krho = 0.5;
79 parameters.Kalpha = 1.5;
80 parameters.Kbeta = -0.6;
81 parameters.backwardAllowed = true;
82 parameters.useConstantSpeed = true;
83 parameters.constantSpeed = 0.1;
84
85 %% Define parameters for Dijkstra and Dynamic Window Approach
86 parameters.dist_threshold= 0.25; % threshold distance to goal
87 parameters.angle_threshold = deg2rad(10);%0.4; % threshold orientation to
       goal
88
89 %%
90 %----------------------- Initialisation of variables
       ----------------------------------------------------------%
91 R = parameters.wheelRadius;
92
93 %% teleoperation program goes here
94
95 v = 0;
96 w = 0;
97
98 K = getkeywaitchar(1);
99 while(K ~= 'q')
100     sprintf('v = %0.1g | w = %0.1g', v, w)
101     switch(K)
102         case 'w'
103             v = v+0.1;
104
105         case 's'
106             v = v-0.1;
107
108
109         case 'a'
110             w = w+0.1;
111
112         case 'd'
113             w = w-0.1;
114
115         case ' '
116             v = 0;
117             w = 0;
118
119     end
120     [phi_l, phi_r] = calculateWheelSpeeds(v, w, parameters);
```

```
    Pioneer_p3dx_setWheelSpeeds(connection, phi_l, phi_r);

    K = getkeywaitchar(1);
end


%% Bring Pioneer_p3dx to standstill
Pioneer_p3dx_setWheelSpeeds(connection, 0.0, 0.0);

if realRobot~= 1
    simulation_stop(connection);
    simulation_closeConnection(connection);
else

end
% msgbox('Simulation ended');
```

Código controlador

```
function [ vu, omega ] = calculateControlOutput( robotPose, goalPose,
    parameters )
%CALCULATECONTROLOUTPUT This function computes the motor velocities for a
    differential driven robot

% 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

vi = parameters.Krho*rho;
wi = parameters.Kalpha*alpha;
```

```
26
27 % Task 4:
28 % parameters.backwardAllowed: This boolean variable should switch the
       between the two controllers
29 % parameters.useConstantSpeed: Turn on constant speed option
30 % parameters.constantSpeed: The speed used when constant speed option is on
31
32 sv = 1;
33
34 if parameters.backwardAllowed
35     if abs(alpha) > pi/2
36         sv = -1;
37         alpha = normalizeAngle(alpha + pi);
38     end
39 end
40
41 v = parameters.Krho*rho;
42 w = parameters.Kalpha*alpha;
43
44 if parameters.useConstantSpeed
45     if rho > parameters.dist_threshold*2
46         w = w*(parameters.constantSpeed/v);
47         v = parameters.constantSpeed*sv;
48     else
49         w = wi;
50         v = parameters.Krho*rho*sv;
51     end
52 end
53
54 vu = v; % [m/s]
55 omega = w; % [rad/s]
56 end
```

Código controle

```
1  %% V-REP Simulation Exercise 4 and 5: Kinematic Control
2  % Tests the implemented control algorithm within a V-REP simulation.
3
4  % In order to run the simulation:
5  %    - Start V-REP
6  %    - Load the scene \EA368 ex2\scene\Exercise2.ttt
7  %    - Hit the run button
8  %    - Start this script
9
10
11 clear;
12 close all;
13 %% Parameters setup
14
```

```
%% define that we will use the real P3DX and/or the simulated one
global realRobot ;
% set realRobot to 1 to use the real robot. For simulation set realRobot to
    0
realRobot=0;

%global ghostPose;
%ghostPose=[0,0,0];

global laserStr;
laserStr = '/perception/laser/';

global poseStr;
poseStr = '/motion/pose';

global vel2Str;
vel2Str = '/motion/vel2';

global stopStr;
stopStr = '/motion/stop';

global parameters;

%% initialization
if realRobot==1
    % http_init will add the necessary paths
    http_init;

    % Declaration of variables
    connection = 'http://10.1.3.130:4950';
    %connection = 'http://143.106.11.166:4950';

    parameters.wheelDiameter = .195;
    parameters.wheelRadius = parameters.wheelDiameter/2.0;
    parameters.interWheelDistance = .381/2;

    %1% Pioneer_p3dx_setTargetGhostPose(connection, -2, 0, 0);

    %% Set the initial pose of the robot ( TO CHANGE INTO A FUNCTION
    SET_POSE)
    %1%http_put([connection '/motion/pose'], struct ('x',0,'y',-2 *1000,'th
    ',90)) ;
    %http_put([connection '/motion/pose'], struct ('x',0,'y',0,'th',0)) ;
    % parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
    connection);
else
    %% Initialize connection with V-Rep
    connection = simulation_setup();
    connection = simulation_openConnection(connection, 0);
```

```
60      simulation_start(connection);
61
62      %% Get static data from V-Rep
63      Pioneer_p3dx_init(connection);
64      parameters.wheelDiameter = Pioneer_p3dx_getWheelDiameter(connection);
65      parameters.wheelRadius = parameters.wheelDiameter/2.0;
66      parameters.interWheelDistance = Pioneer_p3dx_getInterWheelDistance(
        connection);
67      parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
        connection);
68        Pioneer_p3dx_setTargetGhostVisible(connection, 1);
69
70  end
71
72
73  Pioneer_p3dx_setPose(connection, 0,0,0);
74
75  % define target position
76  %Pioneer_p3dx_setTargetGhostPose(connection, 0, -1, deg2rad(90));
77  %1%Pioneer_p3dx_setTargetGhostPose(connection, -1, 0, 0);
78  % Pioneer_p3dx_setTargetGhostPose(connection, 1.6, 0.6, deg2rad(90));
79  % Pioneer_p3dx_setTargetGhostPose(connection, -0.5, 0, deg2rad(180));
80  Pioneer_p3dx_setTargetGhostPose(connection, 3, 0.5, deg2rad(180));
81
82
83  % controller parameters
84  parameters.Krho = 0.5;
85  parameters.Kalpha = 1.5;
86  parameters.Kbeta = -0.6;
87  parameters.backwardAllowed = true;
88  parameters.useConstantSpeed = true;
89  parameters.constantSpeed = 0.1;
90  %% Define parameters for Dijkstra and Dynamic Window Approach
91  parameters.dist_threshold= 0.1; % threshold distance to goal
92  parameters.angle_threshold = deg2rad(5);%0.4; % threshold orientation to
        goal
93
94
95  %% CONTROL LOOP.
96  EndCond = 0;
97
98  while (~EndCond)
99      %% CONTROL STEP.
100     % Get pose and goalPose from vrep
101     [x, y, theta] = Pioneer_p3dx_getPose(connection);
102     [xg, yg, thetag] = Pioneer_p3dx_getTargetGhostPose(connection);
103
104     fprintf('Robot: %0.2g, %0.2g, %0.2g | Ghost: %0.2g, %0.2g, %0.2g\n', x,
        y, theta, xg, yg, thetag)
```

```
105
106     % run control step
107     [ vu, omega ] = calculateControlOutput([x, y, theta], [xg, yg, thetag],
        parameters);
108
109     % Calculate wheel speeds
110     [LeftWheelVelocity, RightWheelVelocity ] = calculateWheelSpeeds(vu,
        omega, parameters);
111
112     % End condition
113     dtheta = abs(normalizeAngle(theta-thetag));
114
115     rho = sqrt((xg-x)^2+(yg-y)^2);  % pythagoras theorem, sqrt(dx^2 + dy^2)
116     EndCond = (rho < parameters.dist_threshold && dtheta < parameters.
        angle_threshold);
117
118     % SET ROBOT WHEEL SPEEDS.
119     Pioneer_p3dx_setWheelSpeeds(connection, LeftWheelVelocity,
        RightWheelVelocity);
120 end
121
122 %% Bring Pioneer_p3dx to standstill
123 Pioneer_p3dx_setWheelSpeeds(connection, 0.0, 0.0);
124
125 if realRobot~= 1
126     simulation_stop(connection);
127     simulation_closeConnection(connection);
128 else
129
130 end
131 % msgbox('Simulation ended');
```