



UNICAMP

Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

IA903 – Introdução à Robótica Móvel

2 de outubro 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	2
2 Teleoperação do robô diferencial	3
3 Controle em malha fechada	3
3.1 Controle ρ e α	3
3.2 Controle ρ , α e β	5
4 Controle melhorado	6
Anexos	9

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).

$$v = \frac{r\varphi_r}{2} + \frac{r\varphi_l}{2} \quad (1)$$

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

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

Para que dada uma velocidade linear e angular desejadas, o robô seja capaz de seguirla, 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.

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

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

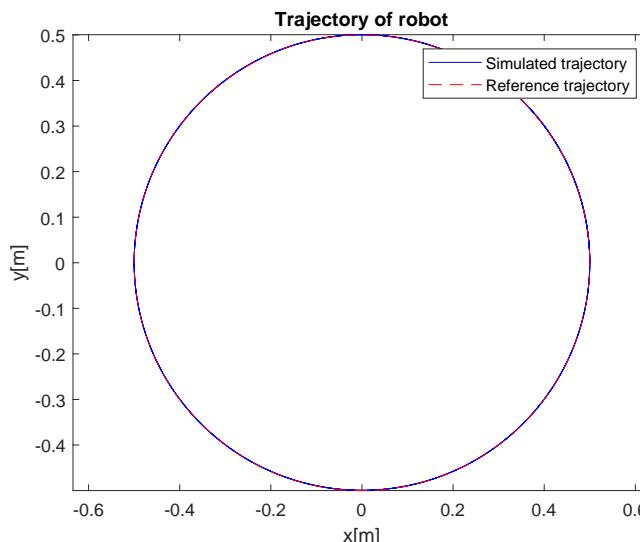


Figura 1: Validação da cinemática inversa do robô.

2 Teleoperação do robô diferencial

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

Algoritmo 1 Algoritmo de teleoperação.

```

1:  $v \leftarrow 0;$ 
2:  $\omega \leftarrow 0;$ 
3:  $K \leftarrow \text{READKEYBOARDKEY}(;)$ 
4: while  $K \neq q$  do
5:   switch  $K$  do
6:     case w:
7:        $v \leftarrow v + 0.1;$ 
8:     case s:
9:        $v \leftarrow v - 0.1;$ 
10:    case a:
11:       $\omega \leftarrow \omega + 0.1;$ 
12:    case d:
13:       $\omega \leftarrow \omega - 0.1;$ 
14:    case SPACE:
15:       $v \leftarrow 0;$ 
16:       $\omega \leftarrow 0;$ 
17:     $[\varphi_l, \varphi_r] \leftarrow \text{CALCULATEWHEELSPEEDS}(v, w, \text{parameters});$ 
18:     $\text{SETWHEELSPEEDS}(\varphi_l, \varphi_r)$ 
19:     $K \leftarrow \text{READKEYBOARDKEY}(;)$ 
20: end while
```

3 Controle em malha fechada

3.1 Controle ρ e α

Aplicado o controle em malha fechada, utilizando apenas o erro de posição e orientação em relação ao ponto de destino, dado por (5), obtém-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.

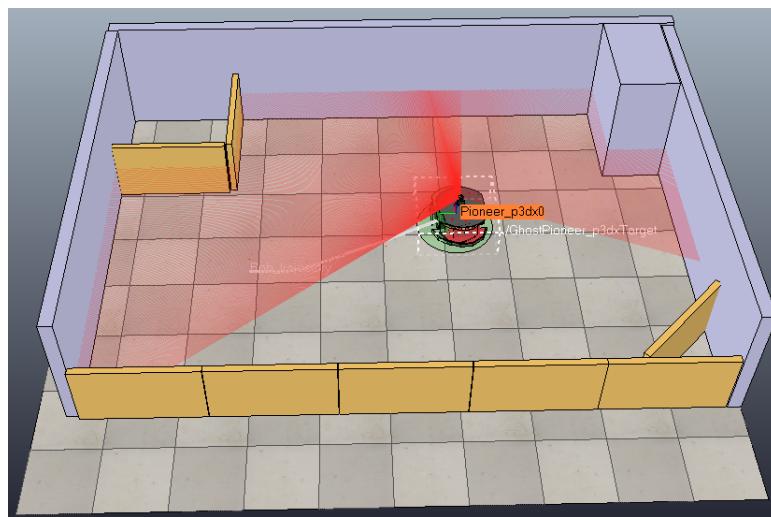


Figura 2: Trajetória do robô para a posição $(1, 6; 0, 6)$ orientado à 90° .

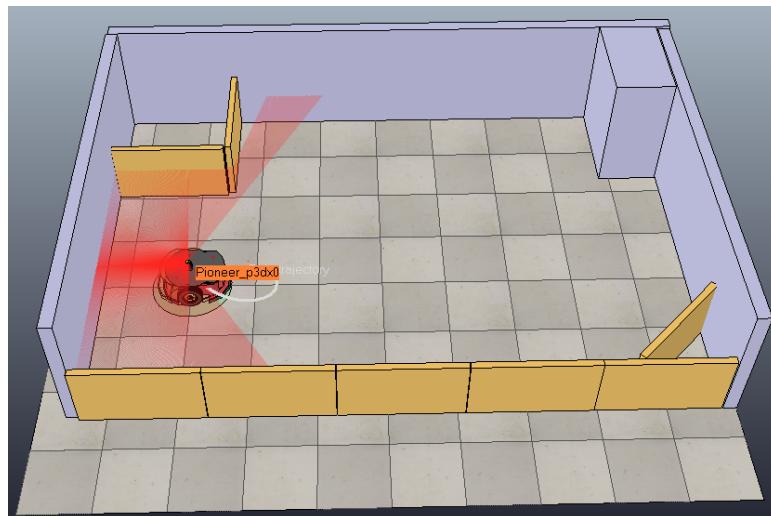


Figura 3: Trajetória do robô para a posição $(-0, 5; 0)$ orientado à 180° .

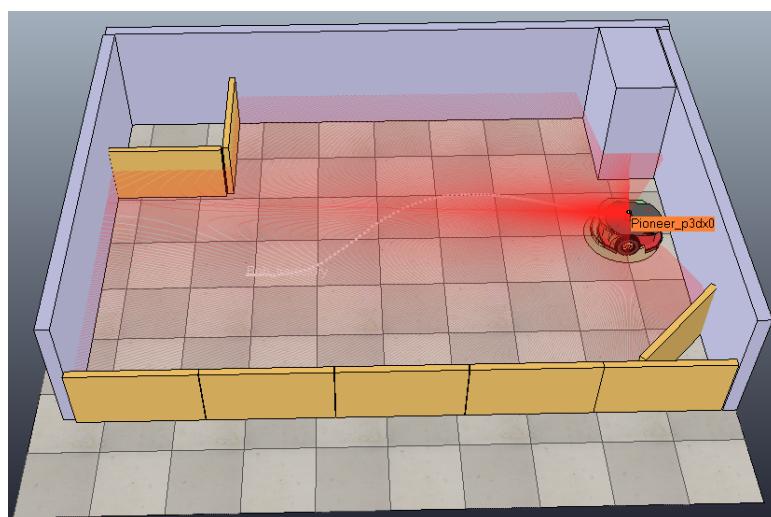


Figura 4: Trajetória do robô para a posição $(3; 0, 5)$ orientado à 180° .

Observa-se que com esse tipo de controle, o robô tende a guiar diretamente até o ponto

de destino, e corrigir sua orientação para a referência desejada uma vez que ele já está no ponto de *target*. Essa estratégia se faz interessante para atingir posições que estão próximas de obstáculos, onde o robô pode chegar pelo caminho que está livre e se posicionar na orientação desejada. Porém, observou-se que, principalmente para o ponto da Figura 2, a trajetória do robô apresenta um grande *overshoot*, realizando desvios desnecessários.

3.2 Controle ρ , α e β

Ampliando o controlador implementado anteriormente, adicionado a parcela de erro do ângulo entre a orientação o vetor do robô ao destino e da orientação de destino, dada por β , se obtém o controle formalizado por (6). Diferentemente do controle anterior, nessa situação, o robô tenta chegar no seu destino com a orientação desejada, evitando que o mesmo precise girar no seu próprio eixo para corrigir a sua orientação. Observa-se na Figura 5, que a trajetória percorrida é mais suave, e leva o robô ao *target* na direção desejada. O mesmo é visto na Figura 6. Porém, na situação da Figura 7, como o alvo está perto da parede, e a orientação do robô é de 180° , o robô colide na tentativa de contornar o destino para aproximar por trás do mesmo, sendo uma situação onde este tipo de controle é falho.

$$\begin{aligned}
 \rho &= \sqrt{(x_g - x)^2 + (y_g - y)^2} \\
 \alpha &= \mod[\text{atan2}(y_g - y, x_g - x) - \theta + \pi, 2\pi] - \pi \\
 v &= k_\rho \rho \\
 \beta &= \mod[\theta_g - \text{atan2}(y_g - y, x_g - x) + \pi, 2\pi] - \pi \\
 \omega &= k_\alpha \alpha + k_\beta \beta
 \end{aligned} \tag{6}$$

Sendo $k_\rho = 0,5$, $k_\alpha = 1.5$ e $k_\beta = -0,6$, respeita-se as condições para estabilidade do sistema.

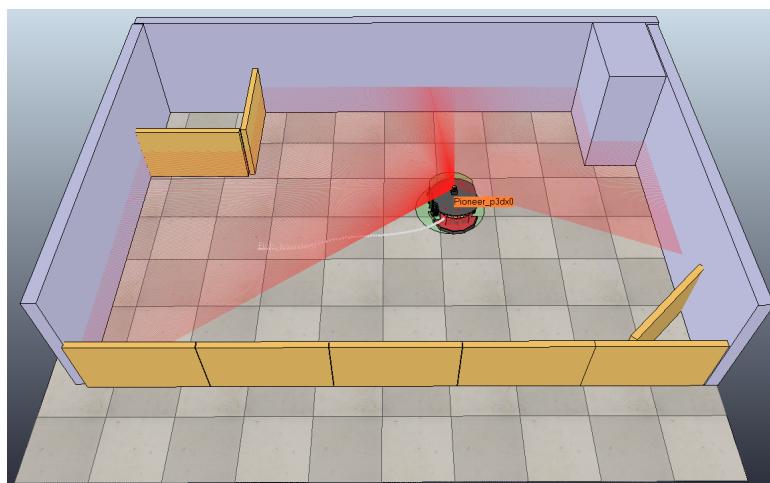


Figura 5: Trajetória do robô para a posição $(1, 6; 0, 6)$ orientado à 90° .

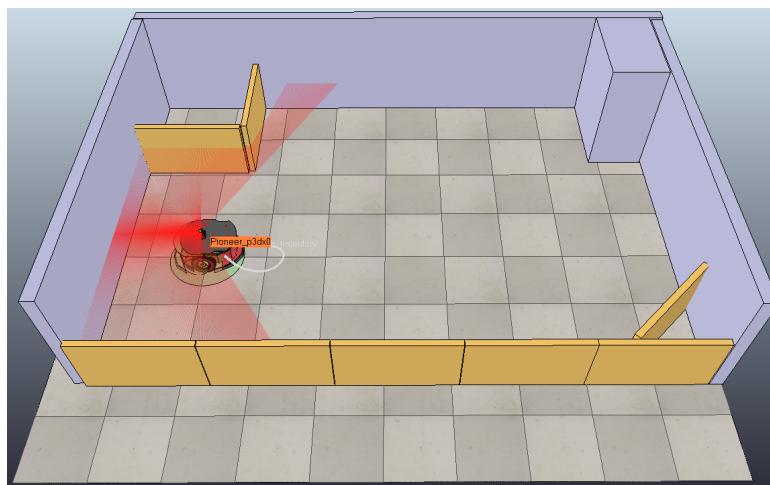


Figura 6: Trajetória do robô para a posição $(-0, 5; 0)$ orientado à 180° .

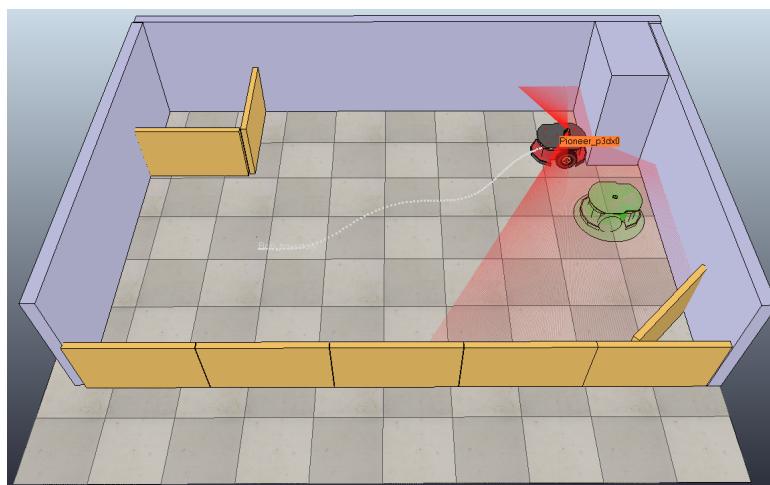


Figura 7: Trajetória do robô para a posição $(3; 0, 5)$ orientado à 180° .

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 acrescido de π , uma vez que o referencial de direção do robô passa a ser $-x$.

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

Algoritmo 2 Algoritmo de direção.

```

1:  $s_v \leftarrow 1;$ 
2: if backwardAllowed then
3:   if  $|\alpha| > \pi/2$  then
4:      $s_v \leftarrow -1;$ 
5:      $\alpha \leftarrow \text{NORMALIZEANGLE}(\alpha + \pi);$ 
6:   end if
7: end if

```

Algoritmo 3 Algoritmo de velocidade.

```

1:  $s_v \leftarrow 1;$ 
2: if useConstantSpeed then
3:   if  $\rho > \text{dist\_threshold} \cdot 2$  then
4:      $\omega \leftarrow \omega \cdot (\text{constantSpeed}/v);$ 
5:      $v \leftarrow \text{constantSpeed} \cdot s_v;$ 
6:   else
7:      $\omega \leftarrow \omega_i;$ 
8:      $v \leftarrow v_i \cdot s_v;$ 
9:   end if
10: end if

```

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 8, 9 e 10.

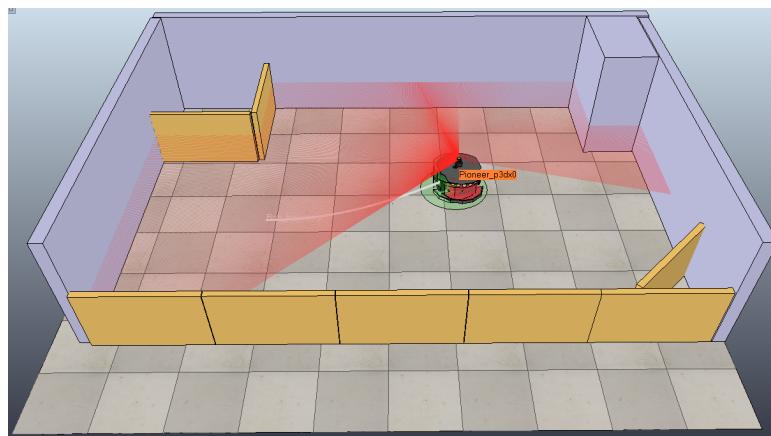


Figura 8: Trajetória do robô para a posição $(1, 6; 0, 6)$ orientado à 90° .

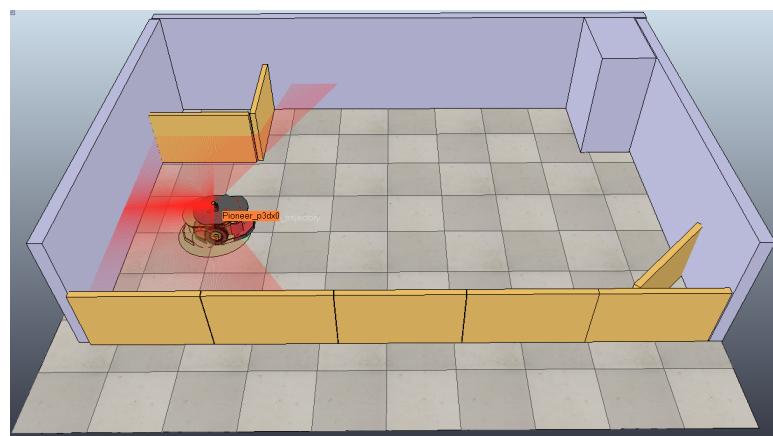


Figura 9: Trajetória do robô para a posição $(-0,5; 0)$ orientado à 180° .

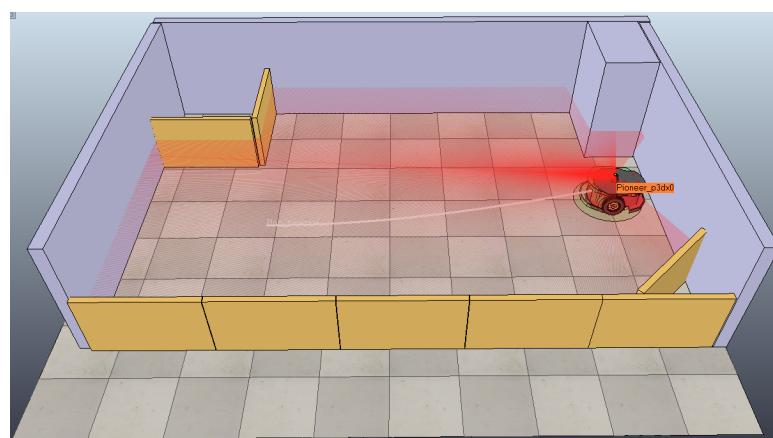


Figura 10: Trajetória do robô para a posição $(3; 0, 5)$ orientado à 180° .

Apêndices

Código cinemática inversa

```
1 function [ LeftWheelVelocity, RightWheelVelocity ] = calculateWheelSpeeds( 
2     vu, omega, parameters )
3 %CALCULATEWHEELSPEEDS This function computes the motor velocities for a
4 % differential driven robot
5
6
7 wheelRadius = parameters.wheelRadius;
8 halfWheelbase = parameters.interWheelDistance/2;
9
10 LeftWheelVelocity = (1/wheelRadius)*vu - (halfWheelbase/wheelRadius)*omega;
11 RightWheelVelocity = (1/wheelRadius)*vu + (halfWheelbase/wheelRadius)*omega;
12
13 end
```

Código teleoperação

```
1 %% V-REP Simulation Exercise 2-3: Teleoperation
2 % Tests the algorithm within a V-Rep simulation.
3
4 % In order to run the simulation:
5 %   - Start V-Rep
6 %   - Load the scene /scene/Exercise2.ttt
7 %   - Hit the run button
8 %   - Start this script
9
10
11 clear;
12 close all;
13 %% Parameters setup
14
15 %% define that we will use the real P3DX and/or the simulated one
16 global realRobot ;
17 realRobot = 0; %realRobot=0 for simulation and realRobot=1 to use the real
18 % robot
19
20 global laserStr;
21 laserStr = '/perception/laser/';
22
23 global poseStr;
24 poseStr = '/motion/pose';
25
26 global vel2Str;
27 vel2Str = '/motion/vel2';
28
29 global stopStr;
30 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
49     %SET_POSE)
50     %1%http_put([connection '/motion/pose'], struct ('x',0,'y',-2 *1000,'th',
51     ',90)) ;
52     %http_put([connection '/motion/pose'], struct ('x',0,'y',0,'th',0)) ;
53     % parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
54     %connection);
55
56 else
57     %% Initialize connection with V-Rep
58     connection = simulation_setup();
59     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(
67     connection);
68     parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
69     connection);
70     Pioneer_p3dx_setTargetGhostVisible(connection, 1);
71
72 end
73 %initialization ends here
74
75 Pioneer_p3dx_setPose(connection, 0,0,0);
76
77 % define target position
78 %Pioneer_p3dx_setTargetGhostPose(connection, 0, -1, deg2rad(90));
79 %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 = %.1g | w = %.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         case 'a'  
109             w = w+0.1;  
110  
111         case 'd'  
112             w = w-0.1;  
113  
114         case ','  
115             v = 0;  
116             w = 0;  
117  
118     end  
119 [phi_l, phi_r] = calculateWheelSpeeds(v, w, parameters);
```

```

121     Pioneer_p3dx_setWheelSpeeds(connection, phi_l, phi_r);
122
123     K = getkeywaitchar(1);
124 end
125
126
127 %% Bring Pioneer_p3dx to standstill
128 Pioneer_p3dx_setWheelSpeeds(connection, 0.0, 0.0);
129
130 if realRobot~= 1
131     simulation_stop(connection);
132     simulation_closeConnection(connection);
133 else
134
135 end
136 % msgbox('Simulation ended');

```

Código controlador

```

1 function [ vu, omega ] = calculateControlOutput( robotPose, goalPose,
2                                                 parameters )
%CALCULATECONTROLOUTPUT This function computes the motor velocities for a
3                                                 differential driven robot
4
5 % current robot position and orientation
6 x = robotPose(1);
7 y = robotPose(2);
8 theta = robotPose(3);
9
10 % goal position and orientation
11 xg = goalPose(1);
12 yg = goalPose(2);
13 thetag = goalPose(3);
14
15 % compute control quantities
16 rho = sqrt((xg-x)^2+(yg-y)^2); % pythagoras theorem, sqrt(dx^2 + dy^2)
17 lambda = atan2(yg-y, xg-x);      % angle of the vector pointing from the
18                                 % robot to the goal in the inertial frame
19 alpha = lambda - theta;          % angle of the vector pointing from the
20                                 % robot to the goal in the robot frame
21 alpha = normalizeAngle(alpha);
22 beta = - lambda + thetag;
23 beta = normalizeAngle(beta);
24
25 % the following parameters should be used:
% Task 3:
% parameters.Kalpha, parameters.Kbeta, parameters.Krho: controller tuning
% parameters

```

```

26 vi = parameters.Krho*rho;
27 wi = parameters.Kalpha*alpha + parameters.Kbeta*beta;
28
29 % Task 4:
30 % parameters.backwardAllowed: This boolean variable should switch the
   between the two controllers
31 % parameters.useConstantSpeed: Turn on constant speed option
32 % parameters.constantSpeed: The speed used when constant speed option is on
33
34 sv = 1;
35
36 if parameters.backwardAllowed
37   if abs(alpha) > pi/2
38     sv = -1;
39     alpha = normalizeAngle(alpha + pi);
40   end
41 end
42
43 v = parameters.Krho*rho;
44 w = parameters.Kalpha*alpha;
45
46 if parameters.useConstantSpeed
47   if rho > parameters.dist_threshold*2
48     w = w*(parameters.constantSpeed/v);
49     v = parameters.constantSpeed*sv;
50   else
51     w = wi;
52     v = parameters.Krho*rho*sv;
53   end
54 end
55
56 vu = vi; % [m/s]
57 omega = wi; % [rad/s]
58 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
15 %% define that we will use the real P3DX and/or the simulated one
16 global realRobot ;
17 % set realRobot to 1 to use the real robot. For simulation set realRobot to
18 % 0
18 realRobot=0;
19
20 %global ghostPose;
21 %ghostPose=[0,0,0];
22
23 global laserStr;
24 laserStr = '/perception/laser/';
25
26 global poseStr;
27 poseStr = '/motion/pose';
28
29 global vel2Str;
30 vel2Str = '/motion/vel2';
31
32 global stopStr;
33 stopStr = '/motion/stop';
34
35 global parameters;
36
37 %% initialization
38 if realRobot==1
39     % http_init will add the necessary paths
40     http_init;
41
42     % Declaration of variables
43     connection = 'http://10.1.3.130:4950';
44     %connection = 'http://143.106.11.166:4950';
45
46     parameters.wheelDiameter = .195;
47     parameters.wheelRadius = parameters.wheelDiameter/2.0;
48     parameters.interWheelDistance = .381/2;
49
50     %1% Pioneer_p3dx_setTargetGhostPose(connection, -2, 0, 0);
51
52     %% Set the initial pose of the robot ( TO CHANGE INTO A FUNCTION
53     %SET_POSE)
54     %1%http_put([connection '/motion/pose'], struct ('x',0,'y',-2 *1000,'th',
55     %,90)) ;
56     %http_put([connection '/motion/pose'], struct ('x',0,'y',0,'th',0)) ;
55     % parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
56     %connection);
56 else
57     %% Initialize connection with V-Rep
```

```
58     connection = simulation_setup();
59     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(
67 connection);
68 parameters.scannerPoseWrtPioneer_p3dx = Pioneer_p3dx_getScannerPose(
69 connection);
70 Pioneer_p3dx_setTargetGhostVisible(connection, 1);
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
93 goal
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,
105             y, theta, xg, yg, thetag)
106
107     % run control step
108     [ vu, omega ] = calculateControlOutput([x, y, theta], [xg, yg, thetag],
109         parameters);
110
111     % Calculate wheel speeds
112     [LeftWheelVelocity, RightWheelVelocity] = calculateWheelSpeeds(vu,
113         omega, parameters);
114
115     dtheta = abs(normalizeAngle(theta-thetag));
116
117     rho = sqrt((xg-x)^2+(yg-y)^2); % pythagoras theorem, sqrt(dx^2 + dy^2)
118     EndCond = (rho < parameters.dist_threshold && dtheta < parameters.
119                 angle_threshold);
120
121
122 %% Bring Pioneer_p3dx to standstill
123 Pioneer_p3dx_setWheelSpeeds(connection, LeftWheelVelocity,
124                               RightWheelVelocity);
125 end
126
127
128 if realRobot~= 1
129     simulation_stop(connection);
130     simulation_closeConnection(connection);
131 else
132
133 end
134 % msgbox('Simulation ended');
```