# Robótica

## Ejercicio 4. Movimiento de un robot diferencial mediante comandos de velocidad

El código adjunto implementa la conducción de una base robótica con sistema diferencial mediante comandos de velocidad u = (v, w), v: velocidad lineal y w: velocidad angular. A partir de este código se pide:

1. Modifique el programa introduciendo una estructura *if-then-else* para que contemple el caso de velocidad angular (w) cero (movimiento en línea recta). En este apartado no existe incertidumbre en el sistema: ni en la pose inicial (matriz $P_{3x3}$), ni en el movimiento (v, w) (matriz $Q_{2x2}$).

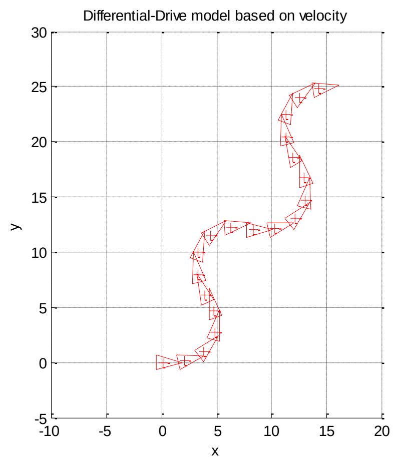

*Figure 1*

In this figure, we see the ideal path of a robot using velocity and angular velocity.

2. Considere ahora error gaussiano en el movimiento: $Q$ = diag ([SigmaV^2 SigmaW^2]);
   a. Introduzca el código que calcule la matriz de covarianza de la pose $P_k$, y dibújela como elipses.

$$P_k = JacFx * P_{k-1} * JacFx' + JacFu * Q * JacFu';$$

   donde $JacFx$ y $JacFu$ son los jacobianos del movimiento con respecto a la pose $x$ y acción $u$. Derive la ecuación de los jacobianos y compruébelas con la solución en los apéndices de las diapositivas.

   b. Dibuje una marca en las poses *ground-truth* generadas aleatoriamente a partir de la Q

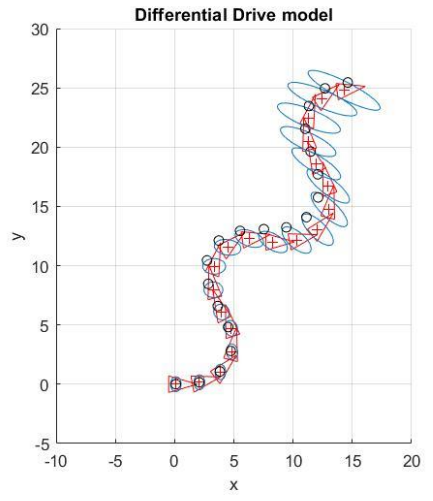

Figure 2

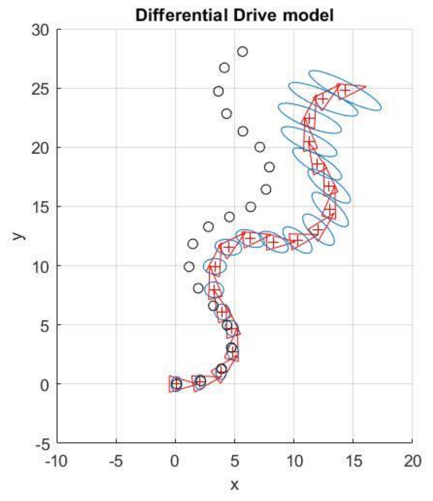

Figure 3

In these figures, we see the ideal path of a robot using velocity and angular velocity with the real position of the robot (represented by a circular point) and an ellipse which represent the possible location where can be the robot. This ellipse follows a gaussian distribution which change in every step of the robot, increasing its size and occupying more space, also changing the sigma.

```
function differential_motion_velocity

clear, close all

% PARAMETERS
dT = 0.1; %time steps size
v = 1; % Linear Velocity
l = 0.5; %Half the width of the robot
SigmaV = 0.1; %Standard deviation of the linear velocity.
SigmaW = 0.1; %Standard deviation of the angular velocity
nSteps =400; %Number of motions


%initial knowledge (prior at k = 0)
x = [0;0;0];
xtrue = x; %Ground-truth position (unknown)
P = diag ([0.2,0.4,0]); %pose covariance matrix 3x3
Q = diag ([SigmaV^2 SigmaW^2]); %motion covariance matrix 2x2

%-------- Set up graphics -----------%
figure (1); hold on; axis equal; grid on; axis ([-20 40 -5 45])
xlabel('x'); ylabel('y');
title ('Differential Drive model');

%-------- Main loop -----------%
for (k = 1: nSteps)
    %control is a wiggle with constant linear velocity
    u = [v; pi/10*sin(4*pi*k/nSteps)];
    R = u (1) /u (2); %v/w Curvature radius

    %jacobians
    sx = sin (x (3)); cx = cos (x (3)); %sin and cos for the previous robot heading
    si = sin (u (2) *dT); ci = cos (u (2) *dT); %sin and cos for the heading
increment
    if (u (2) ==0) %linear motion w=0 --> R = infinite
              %JACOBIAN HERE
              JacF_x= [1 0 -v*sx*dT;
                       0 1 v*cx*dT;
                       0 0 1];
              JacF_u= [cx*dT sx*dT 0;
                       0     0     0];
    else %Non-linear motion w=!0
              %JACOBIAN HERE
              JacF_x= [1 0 R*(-sx*si-cx*(1-ci));
                       0 1 R*(cx*si-sx*(1-ci));
                       0 0 1];
              p1= [cx*si-sx*(1-ci) R*(cx*ci-sx*si);
                   sx*si+cx*(1-ci) R*(sx*ci-cx*si);
                   0               1];
              p2= [1/u (2) -v/ (u (2) ^2);
                   0       dT];
              JacF_u=p1*p2;
    end
    %prediction steps
    %P = JacF_x*P*JacF_x' + JacF_u*Q*JacF_u';
    %xtrue = DifferentialModel (xtrue, u+ [SigmaV; SigmaW]. *randn (2,1), dT);
    x = DifferentialModel (x, u, dT);


    %draw occasionally
    if(mod(k-1,20) ==0)
       DrawRobot(x,'r');
       % PlotEllipse (x, P,0.5);
       % plot (xtrue (1), xtrue(2),'ko');
    end;
end;
```

```
    %------------ MODEL --------------%
function y = DifferentialModel (x, u, dT)
%This function takes pose x and transform it according to the motion u= [v, w]
%applying the differential drive model.
% Dt: time increment
% y: Transformed pose (in world reference)
if (u (2) == 0) %linear motion w=0. Only motion in x
    %dx = u (1) *dT; dy = 0; d_thetha = 0;
    y (1,1) = x (1) + u (1) *dT*cos (x (3));
    y (2,1) = x (2) + u (1) *dT*sin (x (3));
    y (3,1) = x (3);
else %Non-linear motion w=!0
    R=u (1)/u (2); %v/w=r is the curvature radius
    y (1,1) = x (1) - R*sin (x (3)) + R*sin (x (3) +u (2) *dT);
    y (2,1) = x (2) + R*cos (x (3)) - R*cos (x (3) +u (2) *dT);
    y (3,1) = x (3) + u (2) *dT;
end
```