Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional
Unidad Guadalajara

Tarea 8. Control de formación de robots no holónomos con evasión de obstáculos

Presentado por

Jesús Alejandro Díaz Hernández

Presentado para el curso de

Tópicos avanzados de control 2

Curso impartido por: Héctor Manuel Becerra Fermín
Profesor

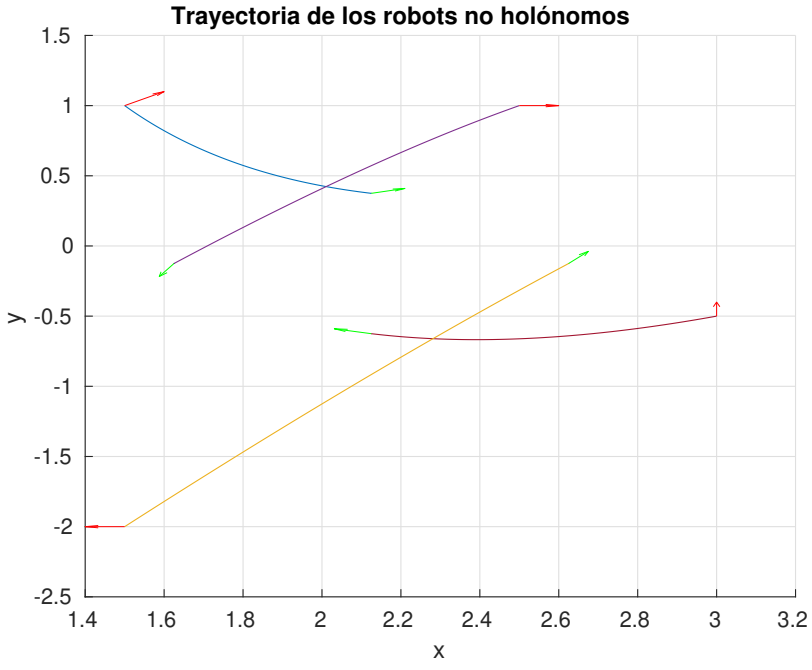
Guadalajara, Jalisco 22 julio 2024

Pregunta 1.-

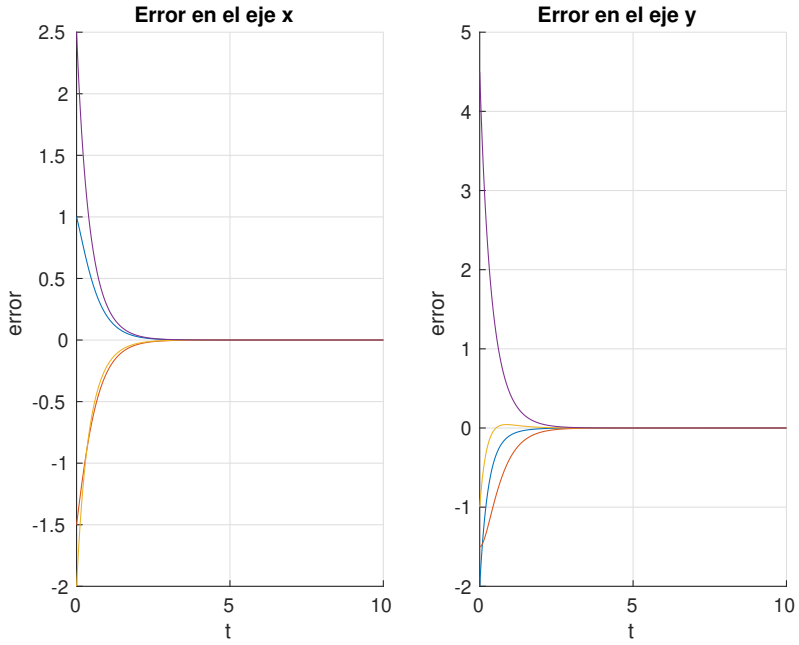
Ajustamos las posiciones iniciales de los robots no holónimos reales para conseguir un cruce en las trayectorias de los agentes como $p_x = \begin{bmatrix} 1 & 2 & 3 & 2 \end{bmatrix}^T$,$p_y = \begin{bmatrix} 1 & 1 & -1 & -2 \end{bmatrix}^T$, A recalcar que se manejan los puntos de control a una distancia de 0.5 unidades del las posiciones iniciales junto con las siguientes orientaciones iniciales dadas en grados $\theta = \begin{bmatrix} 0 & 0 & 90 & 180 \end{bmatrix}$ pensadas para producir un choque entre un par de agentes como se muestra en el siguiente inciso:

Inciso a)

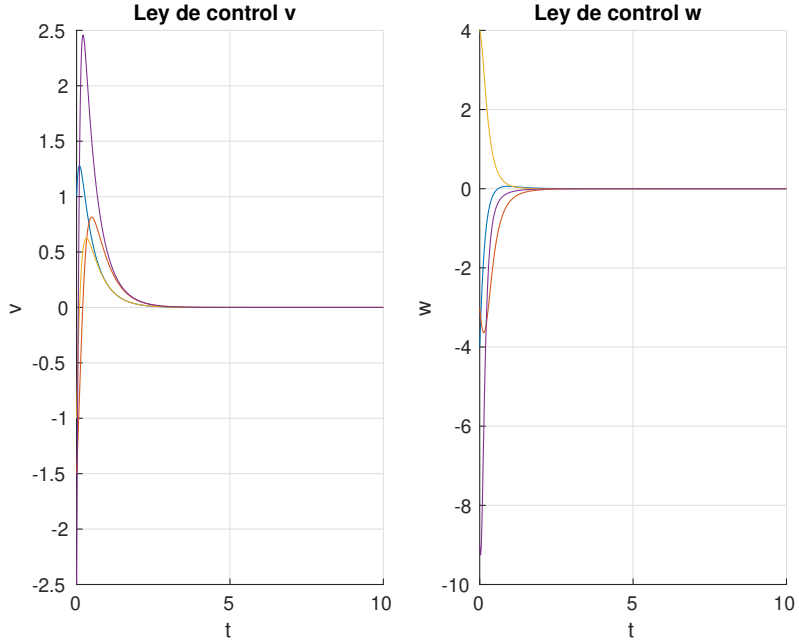
Los inicios de las trayectorias están marcadas con una flecha roja, y los finales con una verde. Como puede notarse la formación de los puntos de control fue especificada como un rombo, como en el artículo de Bernardo, y esta formación se busca en todos los puntos. Es importante mencionar que el hecho de que las trayectorias se crucen no significa que haya un choque, ya que es posible cruzar por el mismo punto, pero en distinto tiempo, sin embargo, se verificó que estos cruces equivalen a choques mediante una simulación paso a paso, así que en esos cruces habría un choque.



La evolución de los errores de consenso con respecto al tiempo para cada coordenada son:



y las leyes de control tanto de la velocidad lineal v y de rotación w:



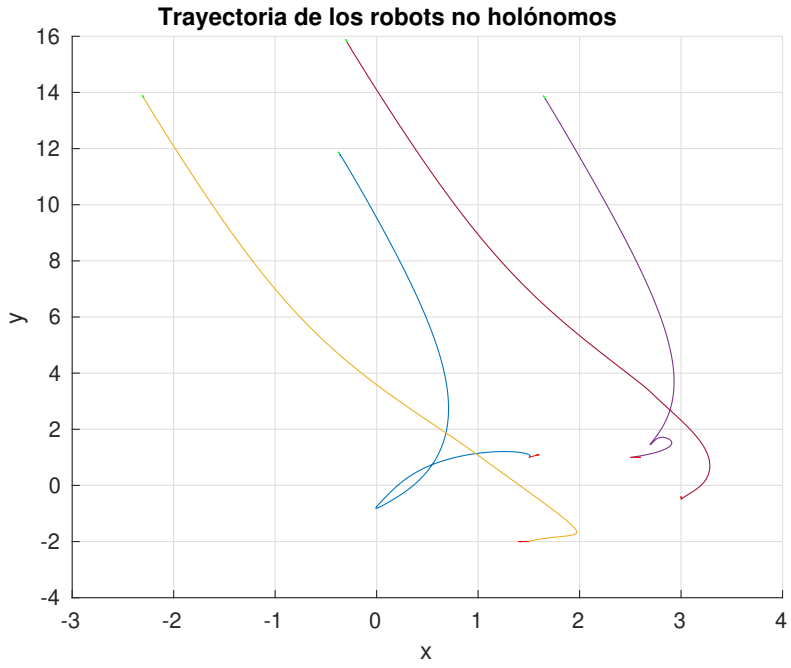
b)

Como la conectividad es no dirigida, el consenso se logra en el promedio de las condiciones iniciales, lo que confirma la gráfica. Se debe tomar en cuenta que se grafican los puntos de control, los que logran la formación, pero los agentes virtuales convergen a $x = 2.125$ y $y = -0.125$

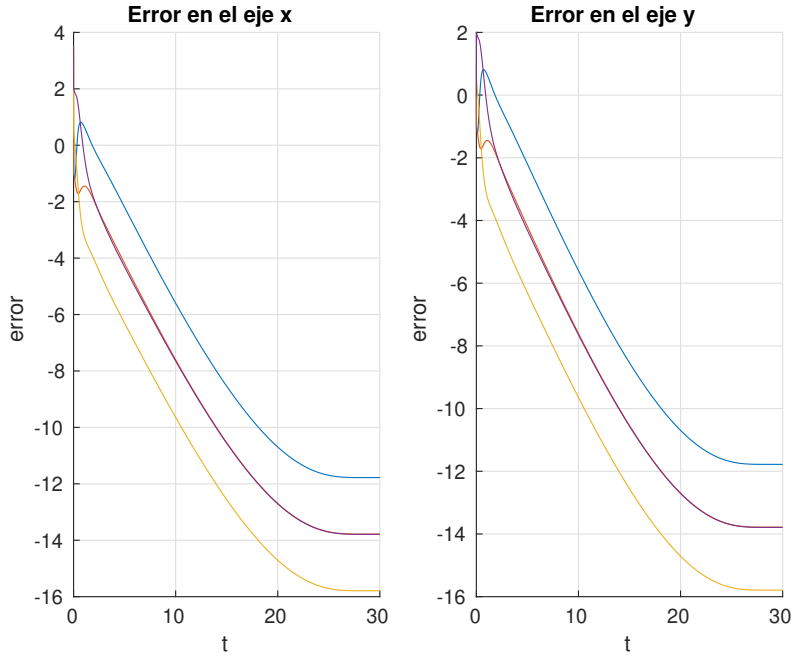
Pregunta 2.-

Implementando el esquema de evasión de obstáculos presentado en Jin&Gans obtenemos los siguientes resultados.

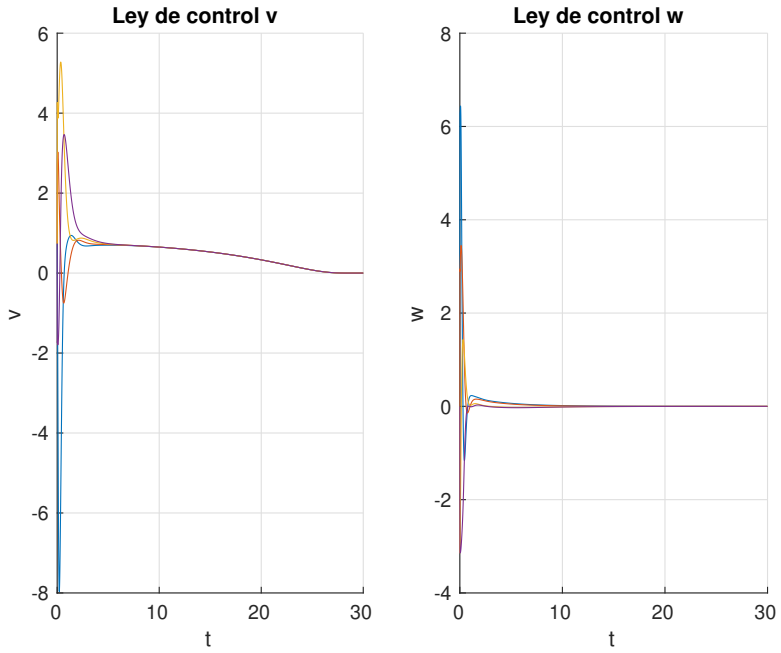
Las trayectorias de los puntos de control son:



Debe recalcarse que aunque se presenten cruces en la formación, se comprobó que no son impactos graficándola a cada paso. Los errores de consenso



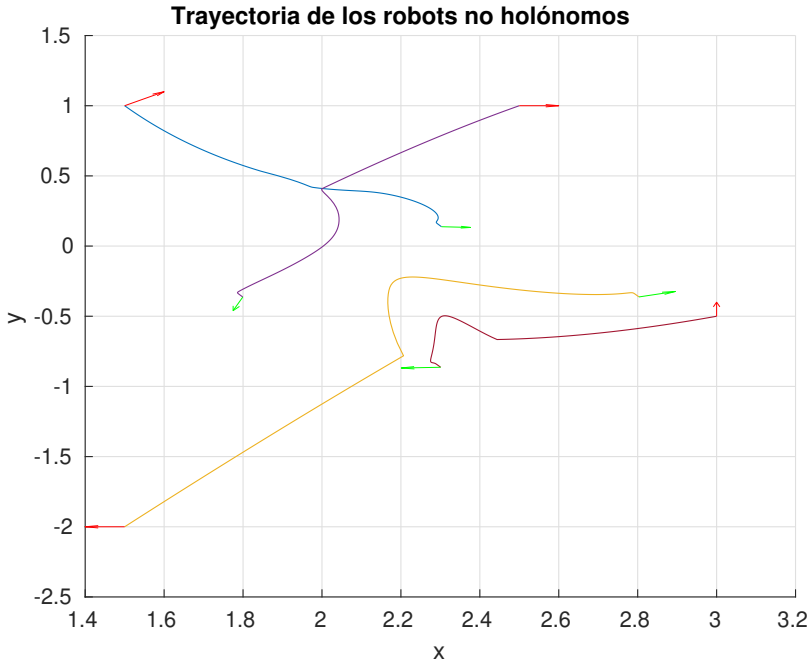
y las leyes de control



Aunque las leyes de control llegan a cero en realidad los agentes no dejan de moverse por lo que realmente no llegan a consenso solo a formación.

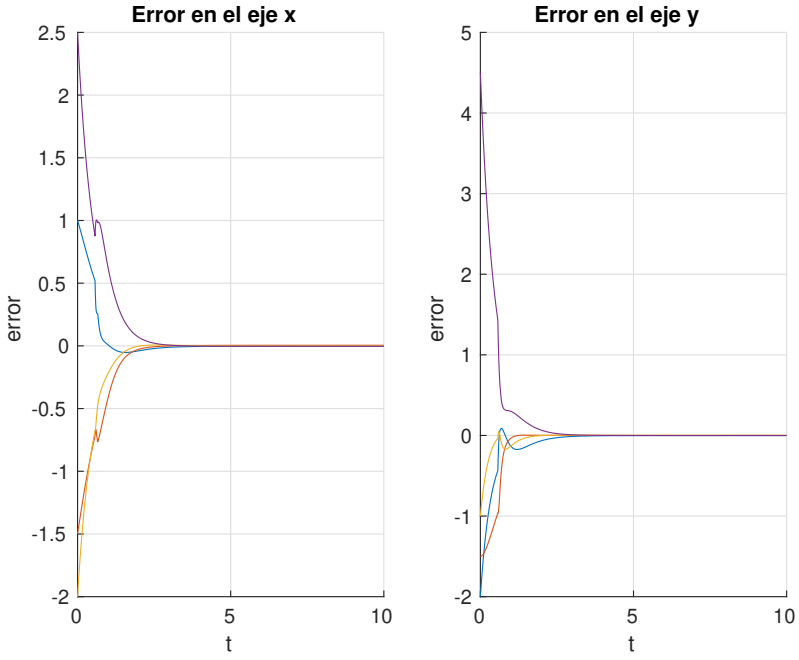
Pregunta 3.-

Implementando el esquema de evasión de obstáculos presentado el artículo de Bernardo Martínez obtenemos los siguientes resultados. Las trayectorias de los puntos de control son:

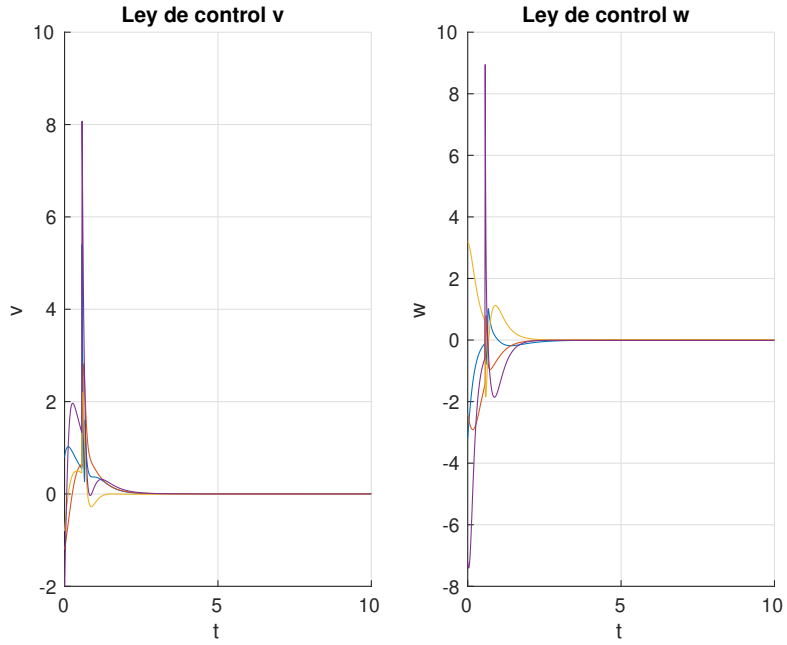


Debe recalcarse que aunque se presenten cruces en la formación, se comprobó que no son impactos graficándola a cada paso.

Los errores de consenso



y las leyes de control



En este punto sí llegan a un valor de consenso, sin embargo, debido a las acciones de evasión no coincide con el promedio de las condiciones iniciales.

Pregunta 4.-

El desarrollo de esquemas de evasión de obstáculos para robots móviles no holónimos es una tarea considerablemente compleja. Aunque se pueda comprender la matemática subyacente, la traducción de estos conceptos teóricos a código práctico no es trivial. La implementación efectiva de estos algoritmos requiere una cuidadosa atención a los detalles y un manejo preciso de los múltiples factores y condiciones que intervienen en la dinámica del sistema.

En particular, la implementación del segundo esquema de evasión de obstáculos, propuesto por Bernardo, mostró cambios bruscos en la ley de control graficada debido a la ausencia del término para transiciones suaves. Por otro lado, el esquema de Jin resultó en una ley de control más suave, aunque presentó dificultades para mantener un estado estacionario y tomó más tiempo en alcanzar la formación deseada.

Anexo (código usado)

Pregunta 1

```
    clc
clearvars
close all

%Parametros iniciales
d=.5;         %Distancia del robot al punto de control
l=0.5;       %Distancia del punto de control al agente virtual
agentes=4;   %Numero de agentes
sigma=0.2;   %Distancia de seguridad
k1=10;      %Ganancia de evasion
k2=2;      %Ganancia de consenso
delta=.1;

%Matriz laplaciana
L=[2 -1  0 -1;
  -1  2 -1  0;
   0 -1  2 -1;
  -1  0 -1  2];
I2=eye(2);
Jr=I2;
kronL=kron(L,I2);

%Posiciones iniciales robots reales, elegidas al azar
x=[1 2 3 2];
y=[1 1 -1 -2];
Theta=deg2rad([0 0 90 180]);
%Posiciones iniciales de los puntos de control
alpha1=[x(1)+d*round(cos(Theta(1))); y(1)+d*round(sin(Theta(1)))];
alpha2=[x(2)+d*round(cos(Theta(2))); y(2)+d*round(sin(Theta(2)))];
alpha3=[x(3)+d*round(cos(Theta(3))); y(3)+d*round(sin(Theta(3)))];
alpha4=[x(4)+d*round(cos(Theta(4))); y(4)+d*round(sin(Theta(4)))];
alphaInit=[alpha1;alpha2;alpha3;alpha4];

%Formacion especificada (un rombo)
z1=l*[round(cos((3*pi)/2)); round(sin((3*pi)/2))];
z2=l*[round(cos(2*pi)); round(sin(2*pi))];
z3=l*[round(cos(pi/2)); round(sin(pi/2))];
z4=l*[round(cos(3*pi)); round(sin(pi))];
z=[z1;z2;z3;z4];

%Posiciones iniciales agentes virtuales
alphav1=alpha1+z1;
alphav2=alpha2+z2;
alphav3=alpha3+z3;
alphav4=alpha4+z4;
alphavInit=[alphav1;alphav2;alphav3;alphav4];

%Condiciones de la simulacion
Dt = 0.01; % Periodo de muestreo
tiempo = 10; % Duracion de la simulacion en segundos
iteraciones = tiempo / Dt;

%Inicializacion de parametros
alphav=zeros(agentes*2,iteraciones+1); %Agentes virtuales
alphav(:,1)=alphavInit;
alpha=zeros(agentes*2,iteraciones);    %Puntos de control
alpha(:,1)=alphaInit;
```

```
theta=zeros(agentes,iteraciones);
theta(:,1)=Theta;
e=zeros(agentes*2,iteraciones);
v_hist=zeros(agentes,iteraciones);
w_hist=zeros(agentes,iteraciones);
%Simulacion
for k=1:iteraciones

    for i=1:agentes
        Id=(2*i)-1:(2*i);

        %Dinamica
        M = [cos(theta(i,k)) -d*sin(theta(i,k));
              sin(theta(i,k))  d*cos(theta(i,k))];
        %Calculo las posiciones del punto de control
        alpha(:,k)=alphav(:,k)-z;
        %Posicion del agente en curso
        alphai=alpha(Id(1):Id(2),k);

        ui=-kronL*alphav(:,k);
        uAplicada=M\ui(Id(1):Id(2),:);


        v=uAplicada(1);
        w=uAplicada(2);

        %Apoximacion de Euler del sistema
        alphav(Id(1):Id(2),k+1)=alphav(Id(1):Id(2),k)+Dt*(M*
            uAplicada);
        theta(i,k+1)=theta(i,k)+Dt*w;

        %Guardo valores para graficar
        e(:,k)=-kronL*alphav(:,k);
        v_hist(i,k)=v;
        w_hist(i,k)=w;
    end
end
```

Pregunta 2

```
clearvars
close all
clc

% Parametros iniciales
l = 0.5; % Distancia del centro del robot al punto de control
% Ganancias del controlador
k1=1;
k2=1;
k3=1;
k4=2;
k0=2;
num_agentes = 4; %Numero de agentes
```

```
Rang=0.1;          %Establece el rango de sensado para todos los
    agentes
ro=Rang;           % Es el rango del obstaculo
kappa=1;           %Esto es para el feed forward term

%Matriz Laplaciana
L=[1   0   0 -1;
  -1   1   0   0;
   0  -1   1   0;
   0   0  -1   1];
I = eye(3);

%Posiciones inciales
x=[1; 2; 3; 2];
y=[1; 1; -1; -2];
Angulo=[0; 0; 90; 180];

% Datos de la simulacion
h = 0.01; % Periodo de muestreo
tiempo = 30; % Duracion de la simulacion en segundos
iteraciones = round(tiempo / h);

%Parametros para la funcion bump
tau = 1;
t0 = 0;            % Tiempo inicial en el que se quiere que inicie
    la funcion
n = tiempo;            % Duracion en segundos durante la cual la
    funcion debe estar activa
bump=zeros(1,iteraciones);

% Inicializar el angulo theta y trayectorias de cada agente
theta = zeros(num_agentes, iteraciones + 1);
x_hist = zeros(num_agentes, iteraciones + 1);
y_hist = zeros(num_agentes, iteraciones + 1);
z_hist = zeros(3 * num_agentes, iteraciones + 1);
q_hist = zeros(3 * num_agentes, 1);
alpha_hist=zeros(3 * num_agentes, iteraciones + 1);

Xi1 = zeros(3 * num_agentes, iteraciones + 1);
qid=zeros(3*num_agentes,iteraciones + 1);
qiu=zeros(3*num_agentes,iteraciones + 1);
qic=zeros(3*num_agentes,iteraciones + 1);
ei=zeros(3,iteraciones + 1);
vi=zeros(num_agentes,iteraciones+1);
wi=zeros(num_agentes,iteraciones+1);
dm=zeros(num_agentes,iteraciones+1);
vl=zeros(num_agentes,iteraciones+1);

%Se declaran vaiables auxiliares donde se almacenaran la diferencia
     de las
%posiciones de los agentes
pi_ox=zeros(num_agentes,num_agentes);
pi_oy=zeros(num_agentes,num_agentes);
norm_pi_o=zeros(num_agentes,num_agentes);
thetaio=zeros(num_agentes,num_agentes);
vthetaio=zeros(num_agentes,num_agentes);
% Se define el parametro de desplazamiento z
```

```
z=1;

%Desplazamientos deseados
 delta= [0; % d41 x
         2; % d41 y
         0;
        -2; % d12 x
         0; % d12 y
         0;
         0; % d23 x
        -2; % d23 y
         0;
         2; % d34 x
         0; % d34 y
         0];

for p = 1:num_agentes
    theta(p, 1) = deg2rad(Angulo(p, 1));
    x_hist(p, 1) = x(p, 1);
    y_hist(p, 1) = y(p, 1);
    % Organizar las posiciones y angulos en el vector qid
    qid(3 * p - 2, 1) = x(p, 1) + l * cos(theta(p, 1));       %
        Posicion en x
    qid(3 * p - 1, 1) = y(p, 1) + l * sin(theta(p, 1));       %
        Posicion en y
    qid(3 * p, 1) = theta(p, 1);       % Angulo theta
    z_hist(3 * p - 2, 1) = x_hist(p, 1) + l * cos(theta(p, 1));
    z_hist(3 * p-1, 1) = y_hist(p, 1) + l * sin(theta(p, 1));
    z_hist(3 * p, 1) = theta(p,1);
    alpha_hist(3*p-2,1)=z_hist(3*p-2,1) + z * cos(theta(p, 1));
    alpha_hist(3 * p-1, 1) = z_hist(3*p-1,1)+z * sin(theta(p, 1));
    alpha_hist(3 * p, 1) = theta(p,1);
end

% Inicializar trayectorias de velocidad
v_hist = zeros(num_agentes, iteraciones);
w_hist = zeros(num_agentes, iteraciones);

% Simulacion
for k = 1:iteraciones

    Xi1(:, k+1) =-k4*kron(L, I)*(alpha_hist(:, k)+delta);
    %Calcular funcion bump en cada iteracion
    t = k * h;  % Definir t en cada iteracion del ciclo como tiempo
         en segundos
    if t0 <= t && t < n + t0
        bump(k) = exp((-tau) /  (1- ((t - t0) / n)^2));
    else
        bump(k) = 0;
    end
    % Sensado de cada agente con respecto a los demas
    for p = 1:num_agentes
        agentes_cercanos = [];
        distancias_cercanas = [];

        for j = 1:num_agentes
            if p ~= j
```

```
121                 % Calcular la diferencia de posicion en x y en y
122                 pi_ox(p,j) = x_hist(p, k) + l * cos(theta(p, k)) -
                        x_hist(j, k)- l * cos(theta(j, k));
123                 pi_oy(p,j) = y_hist(p, k)+ l * sin(theta(p, k)) -
                        y_hist(j, k)- l * sin(theta(p, k));
124                 norm_pi_o(p,j) = sqrt((pi_ox(p,j))^2 + (pi_oy(p,j))
                        ^2);
125                 thetaio(p,j) = atan2(-pi_ox(p,j), -pi_oy(p,j));
126                 vthetaio(p,j) = abs(theta(p,k) - thetaio(p,j));
127
128                 % Verificar si el agente j esta dentro del rango y
                        angulo
129                 if norm_pi_o(p,j) <= dm(p,k) && vthetaio(p,j) < pi
                        /2
130                     agentes_cercanos = [agentes_cercanos; j];
131                     distancias_cercanas = [distancias_cercanas;
                            norm_pi_o(p,j)];
132                 end
133             end
134         end
135         % Determinar el valor de gamma y el agente mas cercano
136         if ~isempty(agentes_cercanos)
137             [min_distancia, idx] = min(distancias_cercanas);
138             agente_mas_cercano = agentes_cercanos(idx);
139             gamma = 1;
140
141             % Tomar las coordenadas del agente mas cercano
142             x_cercano = x_hist(agente_mas_cercano, k)+ l * cos(
                    theta(agente_mas_cercano, k));
143             y_cercano = y_hist(agente_mas_cercano, k)+ l * sin(
                    theta(agente_mas_cercano, k));
144
145             % Asignar el angulo theta_io
146             thetaio(p, agente_mas_cercano) = atan2(-x_cercano, -
                    y_cercano);
147
148             % Asignar el vector p_io
149             p_io = [x_cercano; y_cercano];
150
151             % Buscar un vector ortogonal a p_io
152             p_iorto = [-y_cercano; x_cercano];
153         else
154             gamma = 0;
155             x_cercano = 0; % Valor por defecto si no hay agentes
                    cercanos
156             y_cercano = 0; % Valor por defecto si no hay agentes
                    cercanos
157             p_io = [0; 0]; % Valor por defecto si no hay agentes
                    cercanos
158             p_iorto = [0; 0]; % Valor por defecto si no hay agentes
                     cercanos
159         end
160         qic(1)=p_iorto(1);
161         qic(2)=p_iorto(2);
162         thetaic=atan2(p_iorto(2)-y_hist(p,k),p_iorto(1)-x_hist(p,k)
                );
163         quic(3)=thetaic;
```

```
        % Actualizamos si evadimos o vamos a consenso
        qiu(3 * p - 2, k) = (1 - gamma) * Xi1(3*p-2, k) + gamma *
            qic(1);
        qiu(3 * p - 1, k) = (1 - gamma) * Xi1(3*p-1, k) + gamma *
            qic(2);
        qiu(3 * p, k) = (1 - gamma) *Xi1(3*p, k) + gamma * qic(3);
        %----- En cada iteracion actualiza la distancia que debe
            mantener cada
        %agente para su distancia segura.
        if sin(norm(vthetaio(p,:)))==0
        dm(p,k) = 0;
        else
        dm(p,k) = min(Rang, ro / (sin(norm(vthetaio(p,:)))));
        end
        %---Calculamos la matriz de rotacion-angulo
        R = [cos(theta(p, k)) sin(theta(p, k)) 0;
            -sin(theta(p, k))  cos(theta(p, k)) 0;
            0                        0          1];
        %------- Calcula el error en el marco de referencia local
            con sus
        %vecinos
        ei = R * [qiu(3 * p - 2, k); qiu(3 * p - 1, k);qiu(3 * p, k
            )];

        vl(:,k)=L*vi(:,k);

        if  kappa <= norm(vl(p,k))
            vd(p,k)=k0*exp(-tau);
        else
            vd(p,k)=k0*bump(k);
        end

        vi(p,k) = vd(p,k) * cos(ei(3)) + k1 * ei(1);

        if ei(3)==0
            wi(p,k) = k2 * vd(p,k) * ei(2) + k3 * ei(3);
        else
        wi(p,k) = k2 * vd(p,k) * ei(2) * (sin(ei(3))) / (ei(3)) +
            k3 * ei(3);
        end

        % Dinamica del sistema
        x_hist(p, k + 1) = x_hist(p, k) + h * vi(p,k) * cos(theta(p
            , k));
        y_hist(p, k + 1) = y_hist(p, k) + h * vi(p,k) * sin(theta(p
            , k));
        theta(p, k + 1) = theta(p, k) + h * wi(p,k);

         % Actualizar el vector z con los nuevos valores de x y y
        z_hist(3 * p - 2, k + 1) = x_hist(p, k + 1) + l * cos(theta
            (p, k+1));
        z_hist(3 * p-1, k + 1) = y_hist(p, k + 1) + l * sin(theta(p
            , k+1));
        z_hist(3 * p, k + 1) = theta(p,k+1);
        alpha_hist(3 * p - 2,k+1)=z_hist(3*p-2, k + 1)+ z * cos(
            theta(p, k+1));
```

```
        alpha_hist(3 * p-1,k+1)=z_hist(3*p-1, k + 1)+ z * sin(theta
            (p, k+1));
        alpha_hist(3 * p, k + 1) = theta(p,k+1);
    end
end

figure
hold on
plot(z_hist(1,:),z_hist(2,:))
quiver(z_hist(1, 1), z_hist(2, 1), cos(theta(1,1)), sin(theta(3,1))
    , 0.1, 'r', 'LineWidth', .1, 'MaxHeadSize', 1 );
quiver(z_hist(1, end), z_hist(2, end), cos(theta(1,end)), sin(theta
    (3,end)), 0.1, 'g', 'LineWidth', .1, 'MaxHeadSize', 2);
plot(z_hist(4,:),z_hist(5,:))
quiver(z_hist(4, 1), z_hist(5, 1), cos(theta(2,1)), sin(theta(2,1))
    , 0.1, 'r', 'LineWidth', .1, 'MaxHeadSize', 1 );
quiver(z_hist(4, end), z_hist(5, end), cos(theta(2,end)), sin(theta
    (2,end)), 0.1, 'g', 'LineWidth', .1, 'MaxHeadSize', 2);
plot(z_hist(7,:),z_hist(8,:))
quiver(z_hist(7, 1), z_hist(8, 1), cos(theta(3,1)), sin(theta(3,1))
    , 0.1, 'r', 'LineWidth', .1, 'MaxHeadSize', 1 );
quiver(z_hist(7, end), z_hist(8, end), cos(theta(3,end)), sin(theta
    (3,end)), 0.1, 'g', 'LineWidth', .1, 'MaxHeadSize', 2);
plot(z_hist(10,:),z_hist(11,:))
quiver(z_hist(10, 1), z_hist(11, 1), cos(theta(4,1)), sin(theta
    (4,1)), 0.1, 'r', 'LineWidth', .1, 'MaxHeadSize', 1 );
quiver(z_hist(10, end), z_hist(11, end), cos(theta(4,end)), sin(
    theta(4,end)), 0.1, 'g', 'LineWidth', .1, 'MaxHeadSize', 2);
xlabel('x');
ylabel('y');
title('Trayectoria de los robots no holonomos');
grid on
hold off

t = linspace(0, tiempo, iteraciones);
figure
subplot(1,2,1)
hold on
plot(t,qid(1, 1:iteraciones ) - z_hist(2, 1:iteraciones ))
plot(t,qid(4, 1:iteraciones ) - z_hist(5, 1:iteraciones ))
plot(t,qid(7, 1:iteraciones ) - z_hist(8, 1:iteraciones ))
plot(t,qid(10, 1:iteraciones ) - z_hist(11, 1:iteraciones ))
xlabel('t');
ylabel('error');
title('Error en el eje x');
grid on
hold off
subplot(1,2,2)
hold on
plot(t,qid(2, 1:iteraciones) - z_hist(2, 1:iteraciones))
plot(t,qid(5, 1:iteraciones) - z_hist(5, 1:iteraciones))
plot(t,qid(8, 1:iteraciones) - z_hist(8, 1:iteraciones))
plot(t,qid(11, 1:iteraciones) - z_hist(11, 1:iteraciones))
xlabel('t');
ylabel('error');
title('Error en el eje y');
grid on
```

```
hold off

figure
subplot(1,2,1)
hold on
plot(t,vi(1,1:iteraciones))
plot(t,vi(2,1:iteraciones))
plot(t,vi(3,1:iteraciones))
plot(t,vi(4,1:iteraciones))
xlabel('t');
ylabel('v');
title('Ley de control v');
grid on
hold off
subplot(1,2,2)
hold on
plot(t,wi(1,1:iteraciones))
plot(t,wi(2,1:iteraciones))
plot(t,wi(3,1:iteraciones))
plot(t,wi(4,1:iteraciones))
xlabel('t');
ylabel('w');
title('Ley de control w');
grid on
hold off
```

Pregunta 3

```
clc;
clearvars;
close all;

% Parametros iniciales  tienen que ser mayores a cero
l = 0.5; % Distancia del centro del robot al punto de control
k_x = 1; % Ganancia del controlador para error en x
k_y = 1; % Ganancia del controlador para error en y
k_theta = 1; % Ganancia del controlador para error en theta
kappa = 1; % Constante para el termino feed-forward
n=10;       %Duracion del Bump
tau= 1;    %Constante para el termino feed-forward
R = 1.5; % Rango de deteccion del sensor
r = 1; % Distancia minima aceptable a los obstaculos [x y]
agentes=4; %Numero de agentes

% Laplaciano grafo no dirigido
L = [2 -1  0 -1;
    -1  2 -1  0;
     0 -1  2 -1;
    -1  0 -1  2];
I1 = eye(1);
I2 = eye(2);
%Estos son los Di (Es decir, la raiz del grado de cada nodo
%, pero todos son iguales entonces solo pongo uno)
```

```
Di=[sqrt(2)];
D=Di*eye(agentes);
%Laplaciano normalizado
Ld=inv(D)\L*inv(D); %inv(D)*L*inv(D)

kronp=kron(Ld,I2);
krontheta=kron(Ld,I1);

% Inicializar condiciones iniciales para 4 robots
z = [2;1;1.95; 0.95;3; -1; 2; -2]; % Cada fila es [x, y] de un
    robot
thetaInicial = deg2rad([90, 0, -45, -90]); % Angulos iniciales en
    radianes


% Datos de la simulacion
Dt = 0.01; % Periodo de muestreo
tiempo = 30; % Duracion de la simulacion en segundos
iteraciones = tiempo / Dt;

%inicializo v para poder calcular v_d_i
v_star=zeros(agentes,iteraciones);
w_star=zeros(agentes,iteraciones);

%Incicializo todos los valores
p=zeros(8,iteraciones+1);
p(:,1)=z;
q_star=zeros(12,iteraciones);
q_star(:,1)=[z(1:2);thetaInicial(1);z(3:4);thetaInicial(2);z(5:6);
    thetaInicial(3);z(7:8);thetaInicial(4);];
q_id=zeros(12,iteraciones);
q_ic=zeros(12,iteraciones);
q_iu=zeros(12,iteraciones);
p_io=zeros(8,iteraciones);
p_ia=p_io;
p_ic=p_io;
theta=zeros(4,iteraciones+1);
theta(:,1)=thetaInicial;
theta_io=zeros(4,iteraciones);
theta_ic=zeros(4,iteraciones);
theta_bario=zeros(4,iteraciones);
e=zeros(12,iteraciones);
%HAGO USO DE u_i_star antes de calcularlo entonces lo inicializo en
     zeros
u_i_star=zeros(8,iteraciones+1);
t0_establecido=false;
% Simulacion
for k = 1:iteraciones

      %Aproximacion de Euler de p estrella
      p(:,k+1)=p(:,k)+Dt*(-kronp*p(:,k));

      %Aproximacion de Euler de theta estrella
      theta(:,k+1)=theta(:,k)+Dt*(-krontheta*theta(:,k));

     sump=zeros(8,1);
     % Bucle para calcular la suma de p
```

```
79        for i = 1:agentes
80          for j = 1:agentes
81              % Verificar si existe una conexion entre agentes usando
                    L
82              if L(i, j) == -1
83                  % Indices en sump para el agente i
84                  switch i
85                      case 1
86                          idx_i = 1:2; % Indices para el agente 1 (
                                sump1_x, sump1_y)
87                      case 2
88                          idx_i = 3:4; % Indices para el agente 2 (
                                sump2_x, sump2_y)
89                      case 3
90                          idx_i = 5:6; % Indices para el agente 3 (
                                sump3_x, sump3_y)
91                      case 4
92                          idx_i = 7:8; % Indices para el agente 4 (
                                sump4_x, sump4_y)
93                  end
94                  % Sumar las coordenadas x e y del agente j a las
                        correspondientes del agente i
95                  sump(idx_i) = sump(idx_i) + p(idx_i,k+1);
96              end
97          end
98      end
99
100       sumtheta=zeros(4,1);
101       for i = 1:agentes
102             for j = 1:agentes
103                 %Verifico si existe una conexion entre agentes
                        usando L
104                 if L(i, j) == -1
105                     sumtheta(i) = sumtheta(i)+theta(j,k+1);
106                 end
107             end
108       end
109
110       %Calculo q_id
111       q_id(:,k)=[(Di\sump*inv(Di))' (Di\sumtheta*inv(Di))'];
112       %Lo acomodo para que queden como [x y theta x y theta...]
113       q_id(:,k)=[q_id(1:2,k);q_id(3,k);q_id(4:5,k);q_id(6,k);q_id
              (7:8,k);q_id(9,k);q_id(10:11,k);q_id(12,k)];
114
115
116       %Para calcular el vector de avoidance checo la distancia
              entre
117       %agentes uno por uno
118       for i= 1:agentes
119           switch i
120                       case 1
121                           id_i = 1:2; % Indices para el agente 1
122                       case 2
123                           id_i = 3:4; % Indices para el agente 2
124                       case 3
125                           id_i = 5:6; % Indices para el agente 3
126                       case 4
```

```
                        id_i = 7:8; % Indices para el agente 4
            end
            for j= 1:agentes
                if i~= j
                  switch j
                      case 1
                          id_j = 1:2; % Indices para el agente 1
                      case 2
                          id_j = 3:4; % Indices para el agente 2
                      case 3
                          id_j = 5:6; % Indices para el agente 3
                      case 4
                          id_j = 7:8; % Indices para el agente 4
                  end
                  p_i=p(id_i,k);
                  p_o=p(id_j,k);
                  if norm(p_i-p_o)<r
                      p_io(id_i,k)=p_i-p_o;
                  end
                end
            end
        end

        %Y el bearing angulo lo calculo con la funcin atan2
        for i=1:agentes
            switch i
              case 1
                  id = [1 2]; % Indices para el agente 1 (sump1_x,
                      sump1_y)
              case 2
                  id = [3 4]; % Indices para el agente 2 (sump2_x,
                      sump2_y)
              case 3
                  id = [5 6]; % Indices para el agente 3 (sump3_x,
                      sump3_y)
              case 4
                  id = [7 8]; % Indices para el agente 4 (sump4_x,
                      sump4_y)
            end
          theta_io(i,k)=atan2(-p_io(id(2)),-p_io(id(1)));
        end

        %Calculo el angulo relativo entre el robot y el obstaculo
        theta_bario(:,k)=theta(:,k+1)-theta_io(:,k);

        % Calculo vector atractor
        % Bucle para calcular la suma de p
        for i = 1:agentes
          for j = 1:agentes
              % Verificar si existe una conexion entre agentes usando
                   L
              if L(i, j) == -1
                  % Indices en sump para el agente i
                  switch i
                      case 1
                          idx_i = 1:2; % Indices para el agente 1 (
                              sump1_x, sump1_y)
```

```
                    case 2
                        idx_i = 3:4; % Indices para el agente 2 (sump2_x, sump2_y)
                    case 3
                        idx_i = 5:6; % Indices para el agente 3 (sump3_x, sump3_y)
                    case 4
                        idx_i = 7:8; % Indices para el agente 4 (sump4_x, sump4_y)
                end
                % Sumar las coordenadas x e y del agente j a las correspondientes del agente i
                sump(idx_i) = sump(idx_i) + p(idx_i,k+1);
            end
        end
    end
    p_ia(:,k)=(Di\sump*inv(Di))-p(:,k+1);

    %Calculo el final del vector de avoidance que corresponde a p_ic
    for i=1:agentes
        switch i
            case 1
                id = 1:2; % Indices para el agente 1
            case 2
                id = 3:4; % Indices para el agente 2
            case 3
                id = 5:6; % Indices para el agente 3
            case 4
                id = 7:8; % Indices para el agente 4
        end
        p_ic(id,k)=find_p_ic(p_io(id,k),p_ia(id,k));
    end

    %Calculo el angulo deseado
    for i=1:agentes
        switch i
            case 1
                id = [1 2]; % Indices para el agente 1 (sump1_x, sump1_y)
            case 2
                id = [3 4]; % Indices para el agente 2 (sump2_x, sump2_y)
            case 3
                id = [5 6]; % Indices para el agente 3 (sump3_x, sump3_y)
            case 4
                id = [7 8]; % Indices para el agente 4 (sump4_x, sump4_y)
         end
        theta_ic(i,k)=atan2(p_ic(id(2),k)-p(2,k+1),p_ic(id(1),k)-p(1,k+1));
    end

    %Calculo q_ic y lo acomodo de modo [x_ic,y_ic,theta_ic]
    q_ic(:,k)=[p_ic(1:2,k);theta(1,k);p_ic(3:4,k);theta(2,k);p_ic(5:6,k);theta(3,k);p_ic(7:8,k);theta(4,k)];
```

```
224
225         %Calculo de la funcion gamma para q_iu
226         for i=1:agentes
227           switch i
228               case 1
229                   id = 1:2; % Indices para el agente 1 necesarias
                          para gamma
230                   idq = 1:3;% Indices para el agente 1 para q
231               case 2
232                   id = 3:4; % Indices para el agente 2 necesarias
                          para gamma
233                   idq = 4:6;% Indices para el agente 2 para q
234               case 3
235                   id = 1:2; % Indices para el agente 3 necesarias
                          para gamma
236                   idq = 7:9;% Indices para el agente 3 para q
237               case 4
238                   id = 1:2; % Indices para el agente 4 necesarias
                          para gamma
239                   idq = 10:12;% Indices para el agente 4 para q
240           end
241           gamma=gammaFunc(theta_bario(i,k),p_io(id,k),R,r);
242           q_iu(id,k)=(1-gamma)*q_id(id,k)+gamma*q_ic(id,k);
243         end
244
245         %Genero la M
246         M1=[cos(theta(1,k+1)) 0;
247             sin(theta(1,k+1)) 0;
248             0                 1];
249         M2=[cos(theta(2,k+1)) 0;
250             sin(theta(2,k+1)) 0;
251             0                 1];
252         M3=[cos(theta(3,k+1)) 0;
253             sin(theta(3,k+1)) 0;
254             0                 1];
255         M4=[cos(theta(4,k+1)) 0;
256             sin(theta(4,k+1)) 0;
257             0                 1];
258         M=blkdiag(M1,M2,M3,M4);
259         %Aproximacion de Euler de q estrella
260         q_star(:,k+1)=q_star(:,k)+Dt*(M*u_i_star(:,k));
261
262         %Calculo de los errores por separado
263         e1=[cos(theta(1,k)) sin(theta(1,k)) 0;
264            -sin(theta(1,k)) cos(theta(1,k)) 0;
265             0                0               1]*(q_iu(1:3,k)-q_star
                    (1:3,k+1));
266         e2=[cos(theta(2,k)) sin(theta(2,k)) 0;
267            -sin(theta(2,k)) cos(theta(2,k)) 0;
268             0                0               1]*(q_iu(4:6,k)-q_star
                    (4:6,k+1));
269         e3=[cos(theta(3,k)) sin(theta(3,k)) 0;
270            -sin(theta(3,k)) cos(theta(3,k)) 0;
271             0                0               1]*(q_iu(7:9,k)-q_star
                    (7:9,k+1));
272         e4=[cos(theta(4,k)) sin(theta(4,k)) 0;
273            -sin(theta(4,k)) cos(theta(4,k)) 0;
```

```
274           0                0                1]*(q_iu(10:12,k)-q_star
                  (10:12,k+1));
275
276       %Junto los valores y los guardo para graficarlos
277       e(:,k)=[e1;e2;e3;e4];
278
279       for i=1:agentes
280           switch i
281               case 1
282                 xe=1;ye=2;thetae=3;
283               case 2
284                 xe=4;ye=5;thetae=6;
285               case 3
286                 xe=7;ye=8;thetae=9;
287               case 4
288                 xe=10;ye=11;thetae=12;
289           end
290           for j=1:agentes
291             if abs(v_star(i,k)-v_star(j,k)) >= kappa && i~=j
292                 v_d_i=exp(-tau);
293                 t0_establecido=false;
294             else
295                 t=k*Dt;
296                 if ~t0_establecido
297                     % Establecer t0 la primera vez que entra en el
                            else
298                     t0 = t;
299                     t0_establecido = true;
300                 end
301                 v_d_i=bumpFunc(tau,t,t0,n);
302             end
303           end
304           v_star(i,k)=v_d_i*cos(e(thetae,k))+k_x*e(xe,k);
305           if e(thetae,k)==0
306               operation=1;
307           else
308               operation=(sin(e(thetae,k))/e(thetae,k));
309           end
310           w_star(i,k)=k_y*v_d_i*e(ye,k)*operation+k_theta*e(thetae,
                  k);
311       end
312
313       %Calculo y acomodo u_i_star para graficarlos despues
314       u_i_star(:,k+1)=[v_star(1,k) w_star(1,k) v_star(2,k) w_star
              (2,k) v_star(3,k) w_star(3,k) v_star(4,k) w_star(4,k)]';
315   end
316
317   t = linspace(0, tiempo, iteraciones+1);
318   %
319
320   function p_io_perp = find_p_ic(p_io, p_ia)
321       % Calcular el vector perpendicular
322       p_io_perp = [-p_io(2), p_io(1)];
323
324       % Verificar el producto interno
325       if dot(p_ia, p_io_perp) < 0
```

```
        % Invertir el signo de p_io_perp si el producto interno es
            negativo
        p_io_perp = -p_io_perp;
    end
end

function gamma = gammaFunc(theta_io, p_io, R, r)

    % Calcular el valor absoluto de theta_io
    bar_theta_io = abs(theta_io);

    % Calcular el valor de dm
    if sin(bar_theta_io) == 0
        d_m = R; % Evitar division por cero
    else
        d_m = min(R, r / sin(bar_theta_io));
    end

    % Calcular la norma de p_io
    norm_p_io = norm(p_io);

    % Determinar el valor de gamma
    if bar_theta_io >= pi/2 || norm_p_io > d_m
        gamma = 0;
    else
        gamma = 1;
    end
end

function sigma=bumpFunc(tau,t,t0,n)
    %Se considera que el tiempo nunca sera menor a 0
    if t0 <= t && t < n + t0
        sigma = exp(-tau / (1 - ((t - t0) / n)^2));
    else
        sigma = 0;
    end
end
```