PRÁCTICA 2

Diseño de algoritmos de control clásico para la navegación de robots móviles en entornos ROS-Matlab

El objetivo de esta práctica es diseñar y simular diferentes algoritmos de control para resolver problemas típicos de navegación local de robots móviles, como el control de posición o el seguimiento de trayectorias. Para ello se utilizarán las herramientas de simulación proporcionadas por el entorno ROS.

Esta práctica consta de dos ejercicios:

1. Diseño de una función que implemente el control del robot para alcanzar un punto determinado (destino) dentro del entorno.
2. Diseño de una función que implemente el control del robot para el seguimiento de paredes del entorno a una distancia determinada.

Velocidades máximas:

- Velocidad lineal máxima = 1 m/s
- Velocidad angular máxima = 0,5 rad/s

1. Diseño de un control de posición

Se desea que el robot se desplace desde cualquier posición y orientación hasta una referencia de posición indicada por el usuario. Para realizar pruebas se definirá un entorno libre de obstáculos en ROS de dimensiones 20x20 metros con origen de coordenadas situado en la esquina inferior izquierda del mapa (ver Figura 1).

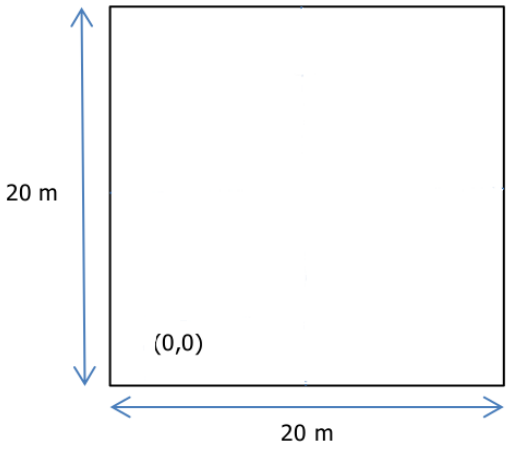


Figura 1. Entorno de movimiento del robot.

Para definir el punto de destino, el usuario introducirá por teclado sus coordenadas (x_p, y_p) en el sistema de referencia del mapa.

1.1. Estructura general del controlador de posición

La Figura 2 muestra la estructura general del sistema de control a desarrollar y las variables que intervienen:

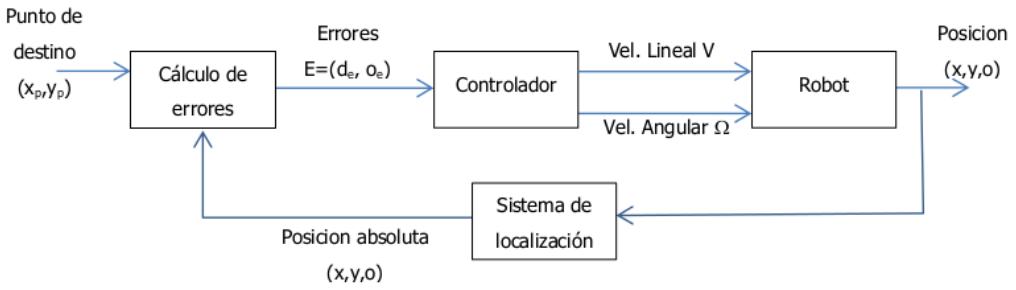


Figura 2. Diagrama de bloques del controlador.

Para la simulación y prueba de este controlador, se utilizarán los ficheros amigobot.launch y practica2.yaml distribuidos con la práctica, y se leerán los datos del sistema de odometría del robot como sistema de localización de este. En una aplicación real más compleja, el sistema de localización de la Figura 2 podría ser un sistema externo (GPS, LPS, etc.) que facilitaría la posición del robot con mayor exactitud.

El lazo de control deberá minimizar el error entre la posición del robot y el punto a alcanzar, actuando convenientemente sobre la velocidad angular y lineal del mismo. Para la implementación y simulación del controlador de la Figura 2 se deben considerar los aspectos siguientes:

- Las salidas del controlador son la velocidad lineal V y angular Ω del robot. Sus entradas serán los errores que se describen a continuación.
- El error de distancia d_e es la distancia euclídea entre la posición actual del robot y la distancia al punto deseado.

$$d_e = \sqrt{(x - x_p)^2 + (y - y_p)^2}$$

- El error de orientación o_e es la diferencia entre el ángulo que une el robot con el punto de destino, y la propia orientación del robot, es decir:

$$o_e = atg\left(\frac{y_p - y}{x_p - x}\right) - o$$

- Cada error afecta únicamente a una de las variables de control: el error de distancia permite ajustar la velocidad lineal (de manera que el robot avanza más despacio cuanto mas cerca está del punto de destino), mientras que el error de orientación modificará la velocidad angular (para que el robot se oriente correctamente hacia el punto de destino). El controlador propuesto se puede dividir en dos controladores independientes como se muestra en la Figura 3.

- Escriba las ecuaciones de los controladores de tipo proporcional mostrados en la Figura 3. El valor de las ganancias de los controladores se ajustará experimentalmente mediante simulación para conseguir un comportamiento adecuado del sistema con una respuesta rápida y sin grandes oscilaciones en la ruta que se siga hacia el punto de destino.

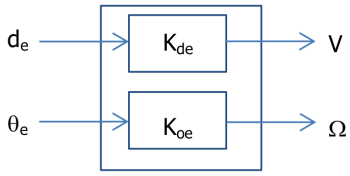


Figura 3. Esquema de controladores P independientes.

1.2. Desarrollo de la práctica

El robot puede estar ubicado inicialmente en cualquier punto del entorno cuya posición se define en el archivo amigobot.launch de la práctica. A continuación, se muestra el esqueleto de la aplicación cliente a desarrollar, en la que el alumno debe rellenar los apartados vacíos para completar el programa:

```
%% INICIALIZACIÓN DE ROS (COMPLETAR ESPACIOS CON LAS DIRECCIONES IP)
setenv('ROS_MASTER_URI','       ');
setenv('ROS_IP','       ');
rosinit() % Inicialización de ROS en la IP correspondiente

%% DECLARACIÓN DE VARIABLES NECESARIAS PARA EL CONTROL

%% DECLARACIÓN DE SUBSCRIBERS
odom = rossubscriber('/robot0/odom'); % Subscripción a la odometría

%% DECLARACIÓN DE PUBLISHERS
pub = rospublisher('/robot0/cmd_vel', 'geometry_msgs/Twist'); %
msg_vel=rosmessage(pub); %% Creamos un mensaje del tipo declarado en
"pub" (geometry_msgs/Twist)

%% Definimos la perodicidad del bucle (10 hz)
r = robotics.Rate(10);
waitfor(r);

%% Nos aseguramos recibir un mensaje relacionado con el robot
while (strcmp(odom.LatestMessage.ChildFrameId,'robot0')~=1)
     odom.LatestMessage
end

%% Umbrales para condiciones de parada del robot
umbral_distancia =
umbral_angulo =

%% Bucle de control infinito
while (1)

%% Obtenemos la posición y orientación actuales
pos=odom.LatestMessage.Pose.Pose.Position;
ori=odom.LatestMessage.Pose.Pose.Orientation;
yaw=quat2eul([ori.W ori.X ori.Y ori.Z]);
yaw=yaw(1);
```

```
%% Calculamos el error de distancia

%% Calculamos el error de orientación

%% Calculamos las consignas de velocidades

consigna_vel_linear =

consigna_vel_ang =

%% Condición de parada

if (Edist<umbral_distancia) && (abs(Eori)<umbral_angulo)
    break;
end

%% Aplicamos consignas de control
msg_vel.Linear.X= consigna_vel_linear;
msg_vel.Linear.Y=0;
msg_vel.Linear.Z=0;
msg_vel.Angular.X=0;
msg_vel.Angular.Y=0;
msg_vel.Angular.Z= consigna_vel_ang;

% Comando de velocidad
send(pub,msg_vel);

% Temporización del bucle según el parámetro establecido en r
waitfor(r);
end

%% DESCONEXIÓN DE ROS
rosshutdown;

end
```

Completado este programa, se pide:

- Observe de qué manera influye el valor de las ganancias del controlador P en el movimiento del robot hacia el punto de destino (valores grandes y valores pequeños). Documéntelo con varios ejemplos y justifique la respuesta.
- Realice experimentos donde se pidan diferentes referencias de posición y documente la influencia de las constantes de control en la respuesta del robot, incluyendo factores como el error de posición en régimen permanente, la presencia de sobreimpulso y la velocidad del robot en llegar al punto deseado.
- Añada al controlador de velocidad angular una parte integral (controlador PI) y documente mediante experimentos las diferencias en las respuestas del controlador P y el controlador PI.
- Compruebe y documente el funcionamiento del controlador realizado en el robot real disponible en el laboratorio. Realice un ajuste de las ganancias del controlador si es necesario.

2. Diseño de un control para el seguimiento de paredes

Con este controlador se pretende guiar el robot para que siga una pared cercana, manteniendo una distancia fija respecto a ella. Las condiciones para la realización de este controlador son las siguientes:

- En este caso debe seleccionarse un entorno que, para facilitar las pruebas, tenga paredes largas para seguir. Para evitar la tarea de creación del bitmap, se sugiere utilizar el mapa de nombre rink.png disponible en la práctica, tomando unas dimensiones para el mismo de 20x20 m, como se muestra en la Figura 4:

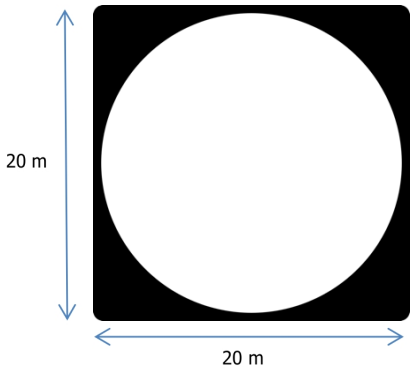


Figura 4. Pared circular.

- En este caso, además de leer la odometría del robot, será necesario tener en cuenta la información de un sensor externo que mida la distancia a la pared. En un primer lugar, se seguirá la pared por el lado izquierdo, utilizándose, por lo tanto, únicamente, la información recibida por el sensor 0 del anillo de ultrasonidos del amigobot.
- El objetivo es que el robot siga de forma reactiva la pared, manteniendo una distancia fija respecto a la misma y una velocidad lineal constante.

2.1. Estructura general del controlador de posición

La Figura 5 muestra la estructura general de este sistema de control.

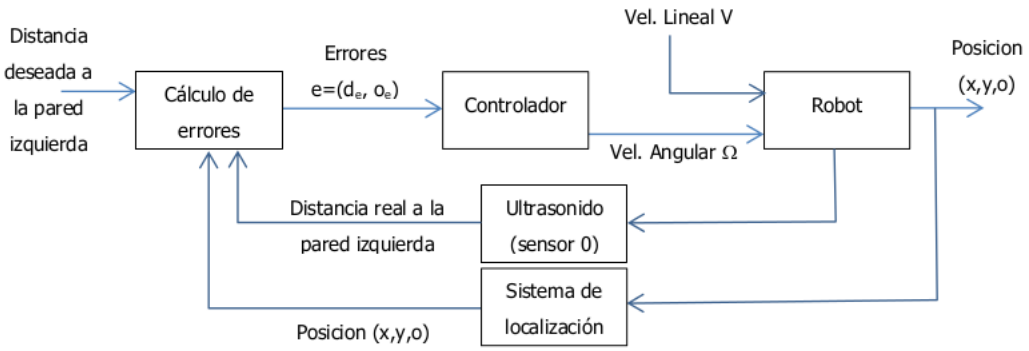
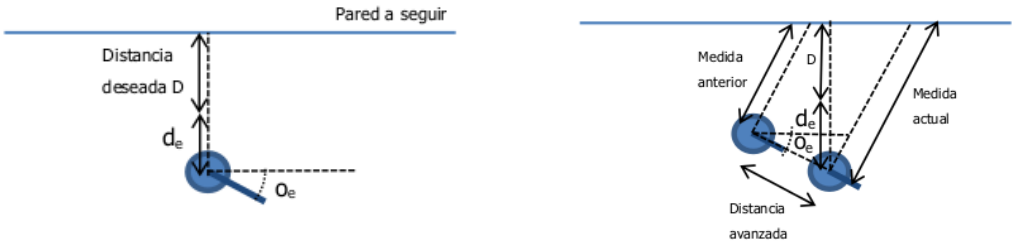


Figura 5. Controlador para seguimiento de pared.

Para el desarrollo de este ejercicio se deben tener en cuenta los siguientes aspectos:

- Aunque la realimentación principal para seguir la pared la ofrece el sensor de ultrasonidos, también es necesario conocer la distancia avanzada por el robot desde la ejecución de la iteración anterior del controlador (para el cálculo del error de orientación, como se verá más adelante). Por eso será necesario utilizar también, como se ha indicado previamente, el sistema de odometría del robot.
- En este caso, el controlador deberá minimizar el error entre la distancia real del robot a la pared y la distancia deseada, así como el error de orientación para que el robot se mantenga paralelo a la pared, actuando convenientemente sobre la velocidad angular Ω del robot. La velocidad lineal del robot se mantendrá constante en todo momento.
- Los errores que se deben minimizar se muestran en las siguientes figuras y se explican detalladamente a continuación:



  - El error de orientación o_e, es la diferencia entre la dirección de la pared y la orientación del robot. Como se observa en la figura, este error puede obtenerse a partir de dos medidas consecutivas realizadas por el sensor 0 del anillo de ultrasonidos y la distancia avanzada por el robot durante ese tiempo (obtenida del sistema de localización):

$$o_e = atan\left(\frac{\text{MedidaSonar}(0)_{\text{actual}} - \text{MedidaSonar}(0)_{\text{anterior}}}{\text{Distancia}_{\text{avanzada}}}\right)$$

  - El error lateral d_e, es la diferencia entre la distancia real del robot a la pared y la distancia deseada D. El sensor 0 del anillo de ultrasonidos nos proporciona la distancia a la pared en la dirección del sensor (menos un offset de 0.105 m, correspondiente a la coordenada 'y' del sensor 0 respecto al centro del sistema de referencia local del robot). Para obtener la distancia del robot a la pared en la dirección perpendicular a la misma hay que corregir la medida multiplicando por el coseno del error de orientación. El error lateral es la distancia del robot al punto de la pared más cercano. Este error se puede calcular utilizando la ecuación de la recta mediante la siguiente expresión:

$$d_e = (\text{MedidaSonar}(0)_{\text{actual}} + 0.105)\, cos(o_e) - D$$

En este caso, el controlador no se puede dividir en dos controladores independientes, ya que los dos tipos de errores sirven para que el controlador ajuste convenientemente la consigna de velocidad angular del robot. La función de transferencia del controlador proporcional sería:

$$\Omega = K_d\ d_e + K_o\ o_e$$

De nuevo, el valor de las ganancias K_d, K_o deberá ser ajustado experimentalmente mediante simulación para conseguir un comportamiento adecuado del sistema, con una respuesta rápida y sin grandes oscilaciones en el seguimiento de la trayectoria.

2.2. Desarrollo de la práctica

Al igual que en el apartado anterior el robot puede estar ubicado inicialmente en cualquier punto del entorno, fuera de la trayectoria a seguir. A continuación, se muestra el esqueleto de la aplicación a desarrollar, en la cual deberán rellenarse los apartados vacíos para completar el programa:

```
%% INICIALIZACIÓN DE ROS (COMPLETAR ESPACIOS CON LAS DIRECCIONES IP)
setenv('ROS_MASTER_URI','       ');
setenv('ROS_IP','       ');
rosinit() % Inicialización de ROS en la IP correspondiente

%% DECLARACIÓN DE VARIABLES NECESARIAS PARA EL CONTROL
MAX_TIME = 1000 %% Numero máximo de iteraciones
medidas = zeros(5,1000);

%% DECLARACIÓN DE SUBSCRIBERS
odom = rossubscriber('/robot0/odom'); % Subscripción a la odometría
sonar0 = rossubscriber('/robot0/sonar_0', rostype.sensor_msgs_Range);

%% DECLARACIÓN DE PUBLISHERS
pub = rospublisher('/robot0/cmd_vel', 'geometry_msgs/Twist'); %
msg_vel=rosmessage(pub); %% Creamos un mensaje del tipo declarado en
"pub" (geometry_msgs/Twist)
msg_sonar0=rosmessage(sonar0);

%% Definimos la periodicidad del bucle (10 hz)
r = robotics.Rate(10);
waitfor(r);

%% Nos aseguramos recibir un mensaje relacionado con el robot
while (strcmp(odom.LatestMessage.ChildFrameId,'robot0')~=1)
     odom.LatestMessage
end

%% Inicializamos variables para el control
i = 0;
pos =
dist =
lastpos =
lastdist =
lastdistav =

%% Bucle de control

while (1)
i = i + 1;
```

```
%% Obtenemos la posición y medidas de sonar
pos=odom.LatestMessage.Pose.Pose.Position;
msg_sonar0 = receive (sonar0);

%% Calculamos la distancia avanzada y medimos la distancia a la pared
  distav =
  dist =

if dist>5
    dist = 5;
end

%% Calculamos el error de distancia y orientación
Eori =
Edist =

medidas(1,i)= dist;
medidas(2,i)= lastdist; %% valor anterior de distancia
medidas(3,i)= distav;
medidas(4,i)= Eori;
medidas(5,i)= Edist;

%% Calculamos las consignas de velocidades
consigna_vel_linear = 0.3
consigna_vel_ang =

%% Condición de parada
if (Edist<0.01) && (Eori<0.01)
    break;
end

%% Aplicamos consignas de control
msg_vel.Linear.X= consigna_vel_linear;
msg_vel.Linear.Y=0;
msg_vel.Linear.Z=0;
msg_vel.Angular.X=0;
msg_vel.Angular.Y=0;
msg_vel.Angular.Z= consigna_vel_ang;

% Comando de velocidad
send(pub,msg_vel);

lastpos = pos;
lastdist = dist;
lastvAng = vAng;
lastdistav = distav;

% Temporización del bucle según el parámetro establecido en r
waitfor(r);

if i==MAX_TIME
    break;
end

end

save('medidas.mat','medidas');

end
%% DESCONEXIÓN DE ROS
rosshutdown;

end
```

Completado este programa, se pide:

- Documente la influencia de las ganancias del controlador en el seguimiento de la trayectoria (valores grandes y valores pequeños). Realice varios experimentos.
- Documente mediante simulaciones el efecto de anular K_d
- Documente mediante simulaciones el efecto de anular K_o
- Documente mediante simulaciones de qué manera influye el valor de la velocidad lineal V del robot en el seguimiento de la trayectoria. Recuerde que el valor de V lo fija el usuario ya que el controlador se encarga de ajustar sólo la velocidad angular.
- Pruebe el controlador realizado en el robot real y documente el funcionamiento de este controlador, identificando los posibles problemas derivados de realizar medidas reales con un sónar.