

PL2- Diseño de algoritmos de control clásico para la navegación de robots móviles en entornos ROS-Matlab

Contenido

1. Resumen	2
2. Introducción	2
3. Sección Principal	3
3.1 Estructura general del controlador de posición	3
3.2 Diseño de un control para el seguimiento de paredes	14
4. Conclusiones	19
5. Bibliografía	19

David Bachiller Vela

Pablo Bermejo Núñez

1. Resumen

Esta práctica es una introducción a la robótica con términos relativamente más avanzados dividida en dos ejercicios en los que se va a tratar en el primero de ellos tanto con el simulador como con el robot real y en el segundo ejercicio solo se va a trabajar con el robot real, en donde tenemos un mapa para cada ejercicio, en el primero es un mapa de 20x20, sin obstáculos en el que el robot se va a mover libremente dada una coordenada destino(x,y), el objetivo principal de esta práctica es utilizar los diferentes sensores que nos proporciona el robot para la toma y análisis de datos que a su vez se harán de forma paralela con el robot real, así podremos comparar y poder sacar conclusiones precisas, mediante graficas o mediante análisis de datos y ver valores como el margen de error, la exactitud, la precisión. En el segundo ejercicio guiar el robot para que siga una pared cercana, manteniendo una distancia fija respecto a ella que 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.

Esta práctica es cierto que nos ha costado menos y nos ha dado menos problemas, ya que por un lado es mas corta que la primera y ya tenemos una serie de conocimientos previos con los diferentes sensores que nos ayudan mucho al desarrollo de esta práctica.

2. Introducción

Para poder realizar esta práctica, lo primero ha sido implementar tanto los nuevos mapas como los nuevos launchers. Tras esto nos hemos puesto a investigar cómo funcionan los sistemas de control que se nos daban en la práctica ya que, a diferencia de la anterior, en este caso los valores de salida afectan a los de entrada.

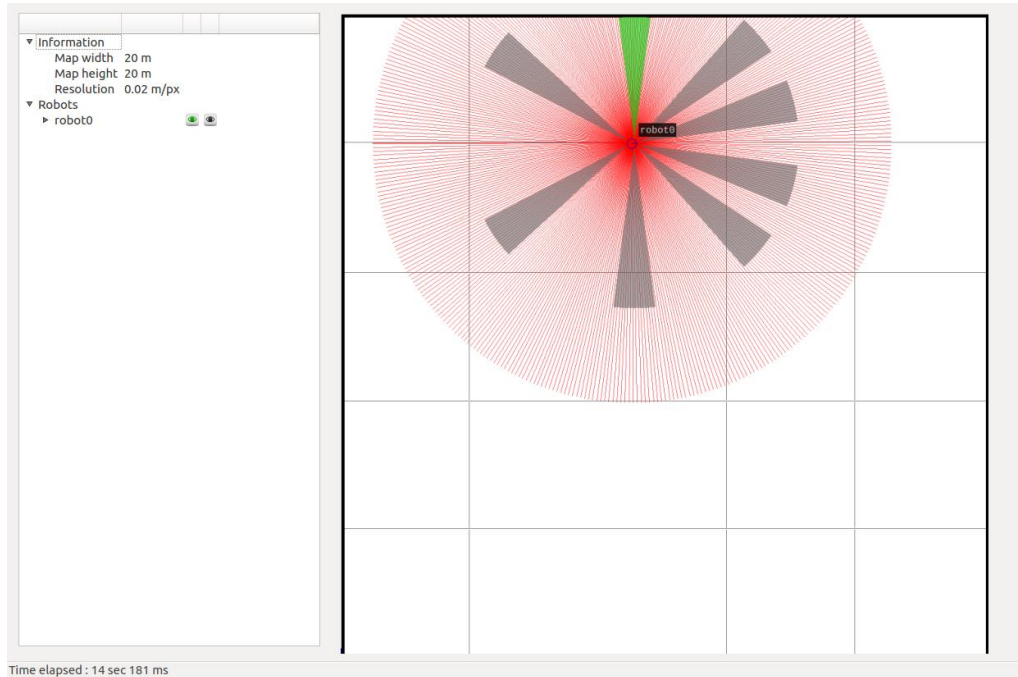
Además de esto, se nos explican unos nuevos conceptos que son los errores tanto de distancia como de orientación. Estos son los encargados de ir actualizando al robot para reorientarlo hacia la posición deseada.

Por último simplemente nos dividimos el trabajo de manera que cada uno pudiese centrarse y especializarse en una parte en concreto para así optimizar tanto el tiempo como los conocimientos. A pesar de esto en todo momento hemos estado cada uno pendiente del otro para aprender todo lo que se nos proponía el enunciado.

3. Sección Principal

3.1 Estructura general del controlador de posición

Se va a trabajar en este mapa de 20x20:



La estructura general del sistema de control a desarrollar y las variables que intervienen:

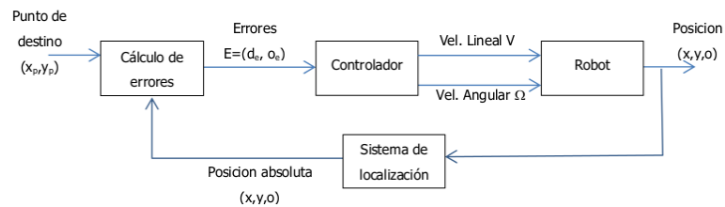


Figura 2. Diagrama de bloques del controlador.

Los diferentes errores de distancia y orientación se calcularán como:

$x \rightarrow x_{\text{destino}}$

$x_p \rightarrow x_{\text{actual}}$

$y \rightarrow y_{\text{destino}}$

$y_p \rightarrow y_{\text{actual}}$

$O \rightarrow \text{yaw}$

- 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:

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

A) 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). Documentelo con varios ejemplos y justifique la respuesta.

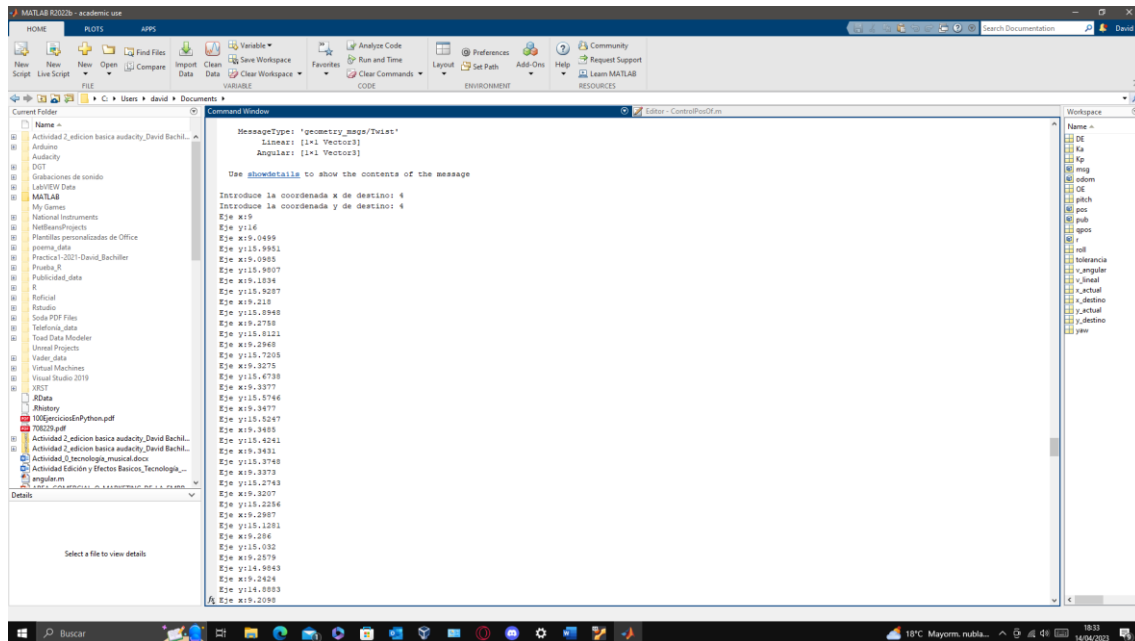
El valor de las ganancias del controlador proporcional (P) tiene un efecto significativo en el movimiento del robot hacia el punto de destino. En general, un valor mayor de P conducirá a una respuesta más rápida y más agresiva del controlador, mientras que un valor menor de P resultará en una respuesta más lenta y más suave.

Por ejemplo, si aumentamos el valor de Kp (en nuestro código `% Constante de proporcionalidad del control de velocidad`) de 0.5 a 1.0 o el de Ka (`% Constante de proporcionalidad del control de orientación`), el robot se moverá hacia el punto de destino con una mayor velocidad lineal y angular, lo que puede hacer que el robot se desplace con una mayor oscilación alrededor del punto de destino y pueda oscilar alrededor del mismo varias veces antes de estabilizarse completamente en el punto de destino. Por otro lado, si disminuimos el valor de Kp a 0.2, el robot se moverá más lentamente y tomará más tiempo en alcanzar el punto de destino, lo que puede resultar en una trayectoria más suave y estable.

En general, un valor de Kp más grande puede llevar a un movimiento más rápido, pero a costa de una mayor oscilación o vibración en la trayectoria. Un valor de Kp más pequeño puede resultar en una trayectoria más suave y estable, pero a costa de una respuesta más lenta. Por lo tanto, el valor de Kp debe elegirse cuidadosamente para equilibrar la velocidad y la estabilidad del movimiento del robot.

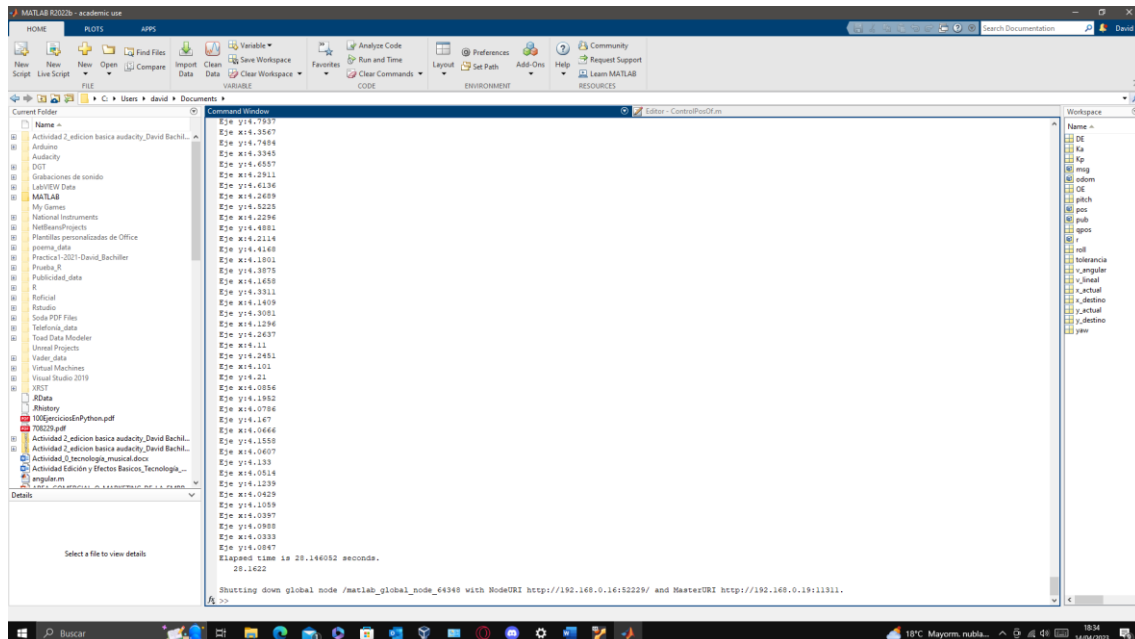
Ej.: El robot avanza con un valor de k_p de 0,2 y de 0,7 hacia las coordenadas (4,4).

Caso $k_p = 0.7$



```
Message: 'geometry_msgs/Twist'  
Linear: [1x1 Vector]  
Angular: [1x1 Vector]  
  
Use showDetails to show the contents of the message  
  
Introduce la coordenada x de destino: 4  
Introduce la coordenada y de destino: 4  
Eje x19  
Eje y14  
Eje x19.2499  
Eje y14.9951  
Eje x19.0965  
Eje y13.9097  
Eje x19.1398  
Eje y13.9287  
Eje x19.218  
Eje y13.9888  
Eje x19.2788  
Eje y13.9121  
Eje x19.2968  
Eje y13.7055  
Eje x19.3275  
Eje y13.6738  
Eje x19.3377  
Eje y13.5746  
Eje x19.3477  
Eje y13.5247  
Eje x19.3685  
Eje y13.4241  
Eje x19.3431  
Eje y13.3748  
Eje x19.3373  
Eje y13.2743  
Eje x19.3207  
Eje y13.2236  
Eje x19.2987  
Eje y13.1281  
Eje x19.286  
Eje y13.032  
Eje x19.2879  
Eje y14.9843  
Eje x19.2424  
Eje y14.0053  
Eje x19.2095
```

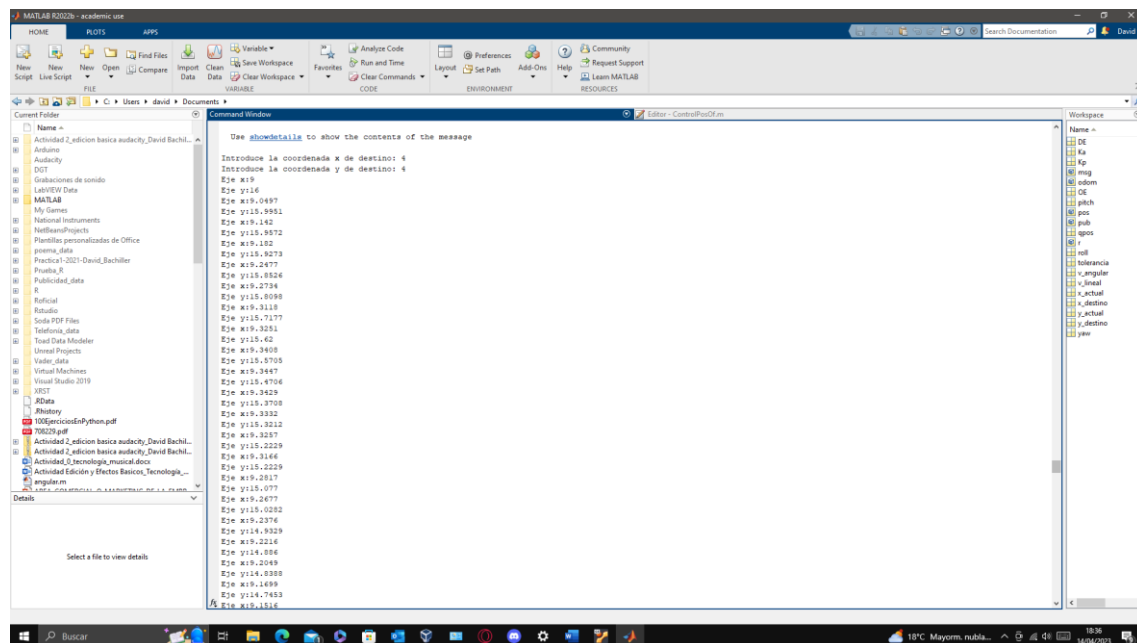
La posición actual del robot es la (9,16) y se va desplazando hasta llegar al punto



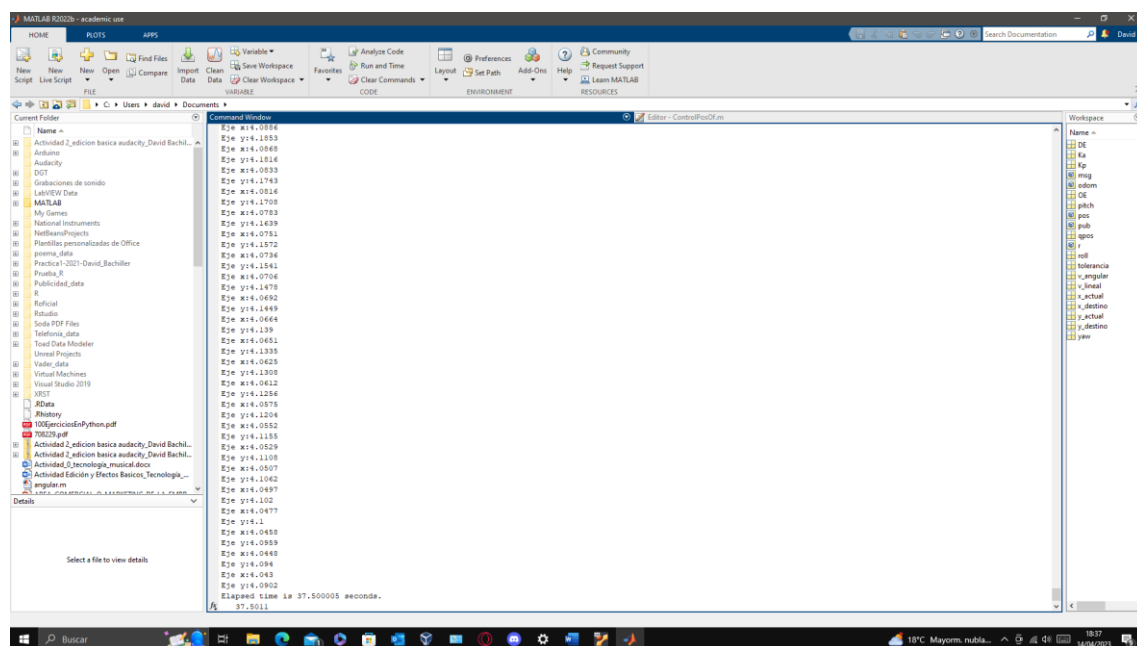
```
Eje y14.7933  
Eje x14.3547  
Eje y14.7484  
Eje x14.3345  
Eje y14.6557  
Eje x14.2911  
Eje y14.6134  
Eje x14.2499  
Eje y14.6223  
Eje x14.2294  
Eje y14.4881  
Eje x14.2114  
Eje y14.4148  
Eje x14.1801  
Eje y14.3875  
Eje x14.1458  
Eje y14.3311  
Eje x14.1458  
Eje y14.3081  
Eje x14.1294  
Eje y14.2437  
Eje x14.11  
Eje y14.2451  
Eje x14.101  
Eje x14.0884  
Eje y14.1952  
Eje x14.0784  
Eje y14.147  
Eje x14.0644  
Eje y14.1559  
Eje x14.0407  
Eje y14.139  
Eje x14.0514  
Eje y14.1239  
Eje x14.0429  
Eje y14.1059  
Eje x14.0397  
Eje y14.0988  
Eje x14.0333  
Eje y14.0847  
Elapsed time is 28.146052 seconds.  
28.1422  
  
Shutting down global node /matlab_global_node_64348 with MasterURI http://192.168.0.16:52229/ and MasterURI http://192.168.0.19:113311.  
R>>
```

El tiempo que tarda es alrededor de 28 segundos y vemos que la precisión es bastante alta en la recogida de datos.

Caso $k_p = 0.2$



La posición actual del robot es la (9,16) y se va desplazando hasta llegar al punto (4,4)

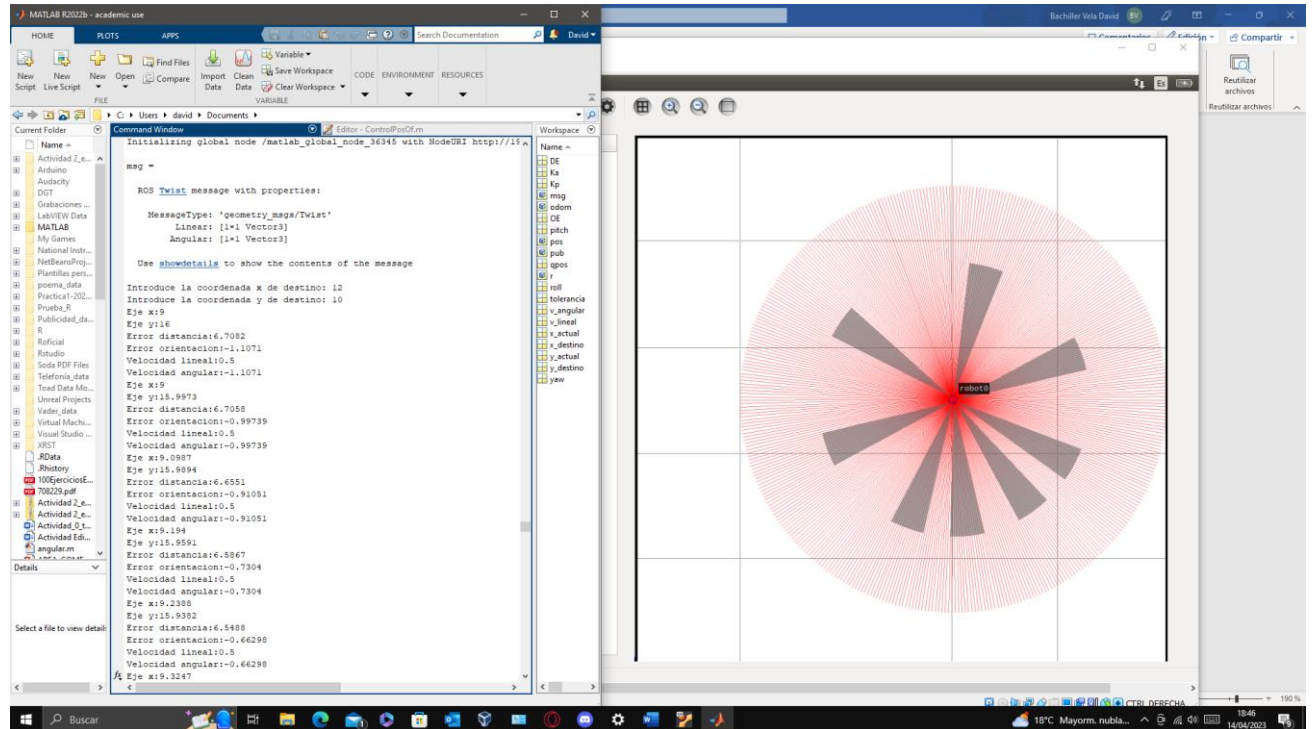


El tiempo que tarda es alrededor de 37 segundos mucho mas elevado que el caso anterior y se puede ver una mayor precisión en la recogida de datos ya que cuando llega a un valor cercano al medio metro de la distancia que se desea llegar disminuye drásticamente la velocidad y recoge mas del doble de datos que con $k_p = 0.7$ lo que nos va a asegurar más precisión y más éxito a la hora de llegar a un punto destino.

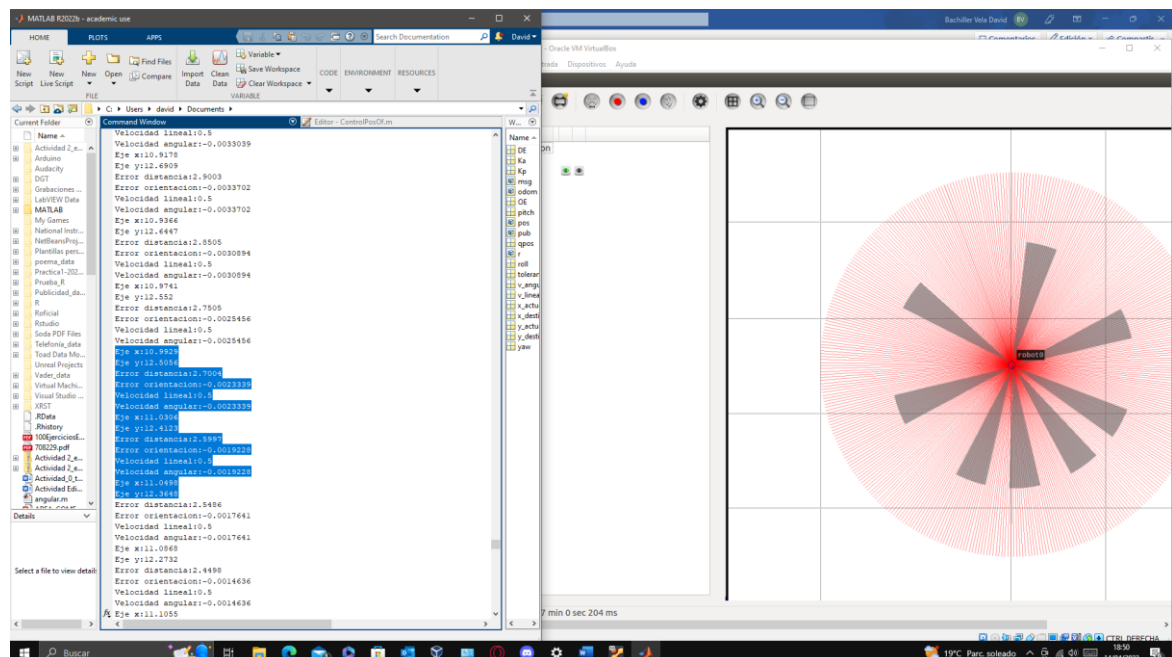
Como decision final hemos decidido coger un $k_p = 0.5$ que nos garantiza un buen tiempo de respuesta y exactitud.

B) 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.

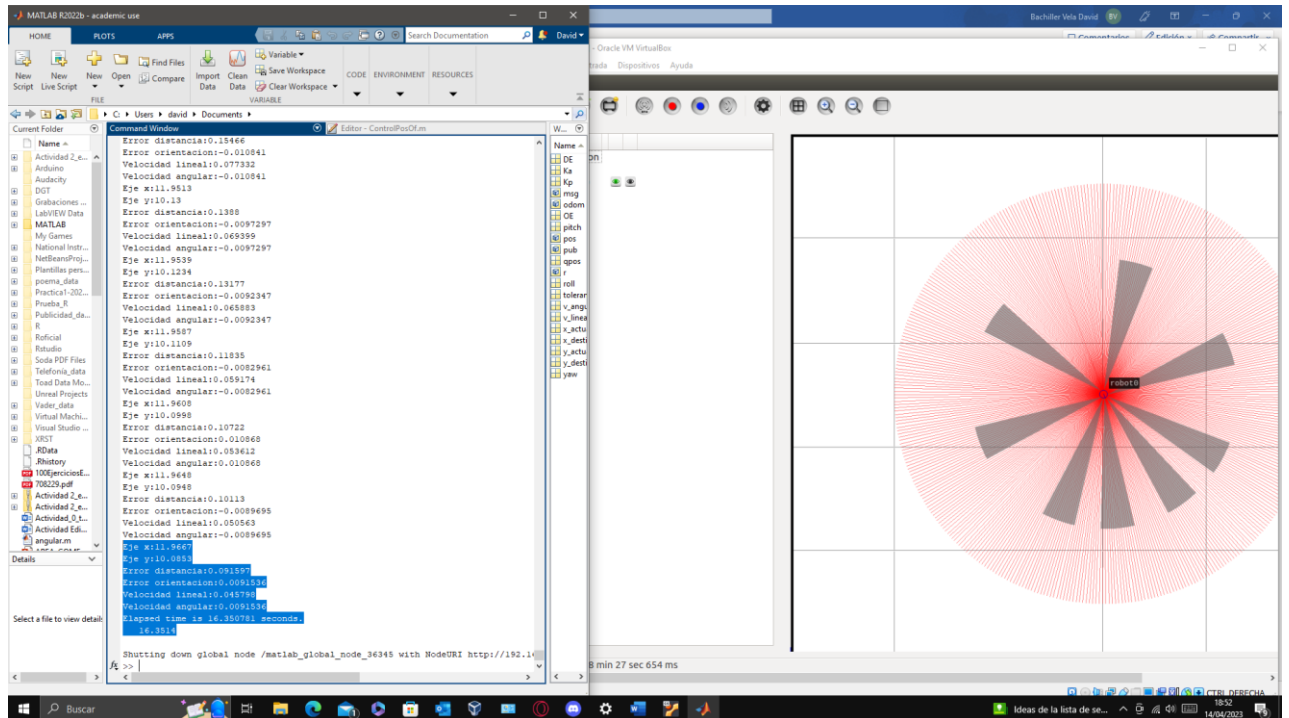
Ej.: Avanzar al punto (12,10)



El robot empieza en la posición (9,16). Los parámetros que se recogen son el Error de distancia, el error de orientación la velocidad lineal y la velocidad angular. A continuación muestro diferentes pantallazos de como van evolucionando estos parámetros hasta llegar al destino, obviamente condicionados por las constantes que proporcionan las diferentes velocidades explicadas en el apartado A



Cada vez vemos como los errores van llegando a 0 por lo que nos vamos aproximando cada vez mas a los valores y la velocidad disminuye drásticamente como se puede ver en las capturas de pantalla proporcionadas.



Eje x:11.9667

Eje y:10.0853

Error distancia:0.091597

Error orientacion:0.0091536

Velocidad lineal:0.045798

Velocidad angular:0.0091536

Elapsed time is 16.350781 seconds.

16.3514

C) Añade 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.

Para añadir una parte integral al controlador de velocidad angular, es necesario integrar el error de orientación en el tiempo. Para ello, hemos utilizado una variable de estado para almacenar la suma acumulada de los errores de orientación (también llamada integral).

```
% Inicializar variables
Kp = 0.5; % Constante de proporcionalidad del control de velocidad
Ki = 0.1; % Constante de integración del control de velocidad
Ka = 1.0; % Constante de proporcionalidad del control de orientación
tolerancia = 0.1; % Tolerancia de distancia y orientación
int_error = 0; % Error integral inicializado en 0

% Bucle de control
while true
    % Obtener la odometría actual del robot
    x_actual = odom.LatestMessage.Pose.Pose.Position.X;
    y_actual = odom.LatestMessage.Pose.Pose.Position.Y;
    pos=odom.LatestMessage.Pose.Pose.Orientation;
    qpos = [pos.W, pos.X, pos.Y, pos.Z];
    % Convertir el resultado anterior en ángulos de Euler en radianes
    [yaw, pitch, roll] = quat2angle(qpos, 'ZYX');

    % Calcular el error de distancia y orientación
    DE = sqrt((x_destino - x_actual)^2 + (y_destino - y_actual)^2);
    OE = atan2(y_destino - y_actual, x_destino - x_actual) - yaw;

    % Control de velocidad lineal en función del error de distancia
    v_lineal = Kp * DE;
    v_lineal = min(v_lineal, 0.5); % Limitar la velocidad máxima

    % Control de velocidad angular en función del error de orientación
    int_error = int_error + OE;
    v_angular = Kp * OE + Ki * int_error;
    v_angular = min(v_angular, 1.0); % Limitar la velocidad angular
    máxima

    % Enviar comandos de velocidad al robot
    msg.Linear.X = v_lineal;
    msg.Angular.Z = v_angular;
    send(pub,msg);

    % Comprobar si se ha llegado a la posición de destino
    if DE < tolerancia && abs(OE) < tolerancia
        break;
    end

    % Esperar un tiempo para evitar saturar el sistema
    pause(0.1);
end
```

```
end

% Parar el robot
msg.Linear.X=0;
msg.Angular.Z=0;
send(pub, msg);
```

En esta modificación, se ha añadido una constante de integración K_i y se ha creado una variable `int_error` que acumula el error integral. El controlador PI calcula la velocidad angular como una suma de la parte proporcional y la parte integral.

Para documentar las diferencias en las respuestas del controlador P y el controlador PI, hemos hecho diferentes pruebas pero en todas hemos mantenido el valor fijo de la constante k_p .

Experimento 1: Movimiento rectilíneo hacia delante

Para ello, se define una posición inicial y una posición objetivo en línea recta frente al robot. Luego, ejecutamos el controlador P y el controlador PI para hacer que el robot se mueva hacia la posición objetivo con una velocidad lineal constante y una velocidad angular controlada por los controladores.

Se repetir el experimento varias veces con diferentes valores de las ganancias del controlador P y PI para comparar las respuestas de ambos controladores. También se puede medir el tiempo que tarda el robot en alcanzar la posición objetivo y la distancia recorrida para comparar la eficiencia de los controladores.

Con este experimento, hemos podido observar cómo el controlador PI mejora la precisión y la velocidad de respuesta en comparación con el controlador P en movimientos rectilíneos.

Otros experimentos que se podrían realizar serían:

Prueba de respuesta transitoria: se puede medir la respuesta del sistema a una entrada de referencia paso. Se puede comparar el tiempo de subida, el tiempo de establecimiento y el error estacionario de los dos controladores.

Prueba de perturbación: se puede aplicar una perturbación al sistema, como un empujón en una dirección, y medir la respuesta del sistema. Se puede comparar la capacidad de los dos controladores para resistir las perturbaciones y recuperarse de ellas.

Prueba de rechazo a la perturbación: se puede aplicar una perturbación periódica al sistema y medir la respuesta del sistema. Se puede comparar la capacidad de los dos controladores para rechazar la perturbación y mantener una velocidad angular constante.

PD: Estas tres ultimas no las he llegado a probar solo las comento como una posible idea.

D) 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.

Primeramente, como en las practicas anteriores ahí que hacer los pertinentes cambios en el código para poder conectarnos al robot real.

```
%% INICIALIZACIÓN DE ROS
% Se definen las variables de entorno ROS_MASTER_URI (ip del Master) y ROS_IP (IP
%de la máquina donde se ejecuta Matlab). Si se está conectado a la misma red, la
%variable ROS_IP no es necesario definirla.
setenv('ROS_MASTER_URI','http://172.29.30.172:11311') %Aqui poner la ip del robot
setenv('ROS_IP','172.29.29.59') %Aqui poner la ip de windows
rosinit % Inicialización de ROS

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

%% DECLARACIÓN DE PUBLISHERS
pub_enable=rospublisher('/cmd_motor_state','std_msgs/Int32');
pub = rospublisher('/cmd_vel', 'geometry_msgs/Twist'); %
%declaración mensaje
msg_enable_motor=rosmessage(pub_enable);
%activar motores enviando enable_motor = 1
msg_enable_motor.Data=1;
msg=rosmessage(pub) %% Creamos un mensaje del tipo declarado en "pub (geometry_msgs/Twist)
send(pub_enable,msg_enable_motor);
```

Esto ya esta comentado y detallado en la PL1.

Para esta parte hemos hecho múltiples pruebas con el robot y debido al poco espacio que disponíamos y la prueba consiste en que el robot vaya a la posición (2,2) y vuelva a la posición origen (0,0).

PD: TODO ESTO SE PODRA VER DE FORMA DETALLADA EN EL VIDEO DISPONIBLE EN LA CARPETA DE LA PRACTICA

El objetivo de esta parte es comparar los diferentes márgenes de error del robot real respecto a la simulación que tras hacer las pruebas hemos comprobado que el error es mas grande en el robot real debido al error acumulado de la odometría por lo que vamos lastrando un pequeño error que con el simulador no pasa.

Inicialmente lo tenemos colocado en la posición (0,0):

```
Use showdetails to show the contents of the message

Introduce la coordenada x de destino: 2
Introduce la coordenada y de destino: 2
Eje x:0
Eje y:0
Error distancia:2.8284
Error orientacion:0.7854
Velocidad lineal:0.5
Velocidad angular:0.7854
Eje x:0
Eje y:0
Error distancia:2.8284
Error orientacion:0.7854
Velocidad lineal:0.5
Velocidad angular:0.7854
Eje x:0
Eje y:0
Error distancia:2.8284
Error orientacion:0.7854
Velocidad lineal:0.5
Velocidad angular:0.7854
Eje x:0.007
Eje y:0
Error distancia:2.8235
Error orientacion:0.7697
Velocidad lineal:0.5
Velocidad angular:0.7697
Eje x:0.027
Eje y:0
```

El robot va reduciendo poco a poco el error y la velocidad cuando se acerca al punto.

```
Velocidad angular:0.0093894
Eje x:1.806
Eje y:1.802
Error distancia:0.2772
Error orientacion:0.010204
Velocidad lineal:0.1386
Velocidad angular:0.010204
Eje x:1.836
Eje y:1.831
Error distancia:0.23549
Error orientacion:0.015014
Velocidad lineal:0.11775
Velocidad angular:0.015014
Eje x:1.87
Eje y:1.865
Error distancia:0.18742
Error orientacion:0.018866
Velocidad lineal:0.093708
Velocidad angular:0.018866
Eje x:1.889
Eje y:1.885
Error distancia:0.15983
Error orientacion:0.00024397
Velocidad lineal:0.079916
Velocidad angular:0.00024397
Eje x:1.907
Eje y:1.903
Error distancia:0.13438
Error orientacion:0.0035962
Velocidad lineal:0.06719
fx Velocidad angular:0.0035962
<
```

```
Eje x:1.936
Eje y:1.938
Error distancia:0.089107
Error orientacion:-0.033325
Velocidad lineal:0.044553
Velocidad angular:-0.033325
Elapsed time is 8.192728 seconds.
8.1938
```

Llega al punto 1.936 en el eje x y al punto 1.938 en el eje y y por lo que el margen de error es aparentemente igual que en el simulador el error crece mas cuando le vuelvo a mandar que vuelva a la posición origen.

```
Velocidad angular:-0.033325
Elapsed time is 8.192728 seconds.
8.1938

Introduce la coordenada x de destino: 0
Introduce la coordenada y de destino: 0
Eje x:1.972
Eje y:1.969
Error distancia:2.7867
Error orientacion:-3.1249
Velocidad lineal:0.5
Velocidad angular:-3.1249
Eje x:1.978
Eje y:1.975
Error distancia:2.7952
Error orientacion:-3.1249
Velocidad lineal:0.5
Velocidad angular:-3.1249
Eje x:1.984
Eje y:1.981
```

El robot gira para colocarse de vuelta a la posición (0,0) y como se puede ver en las imágenes al hacer el giro avanza incluso hasta 2,5 recorriendo medio metro de error acumulado de

edometría que luego al llegar al punto (0,0), vemos como el robot esa como 40 cm – 50 cm colocado a la izquierda de donde lo habíamos mandado inicialmente, esto se podrá ver en el video de forma mas detallada.

```
Velocidad angular:-0.00034  
Eje x:2.716  
Eje y:1.137  
Error distancia:2.9444  
Error orientacion:-0.12714  
Velocidad lineal:0.5  
Velocidad angular:-0.12714  
Eje x:2.557  
Eje y:1.018  
Error distancia:2.7522  
Error orientacion:-0.14471  
Velocidad lineal:0.5  
Velocidad angular:-0.14471  
Eje x:2.422  
Eje y:0.952  
Error distancia:2.6024  
Error orientacion:-0.026913
```

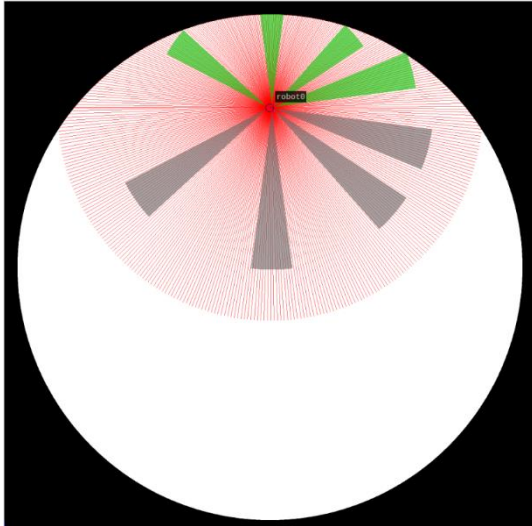
Y ya avanza hasta llegar al (0,0)

```
Velocidad angular:-0.00034  
Eje x:0.097  
Eje y:0.05  
Error distancia:0.10913  
Error orientacion:0.03961  
Velocidad lineal:0.054564  
Velocidad angular:0.03961  
Eje x:0.084  
Eje y:0.044  
Error distancia:0.094826  
Error orientacion:0.046181  
Velocidad lineal:0.047413  
Velocidad angular:0.046181  
Elapsed time is 11.502322 seconds.  
11.5026
```

3.2 Diseño de un control para el seguimiento de paredes

En este apartado se nos pide crear un código que haga que nuestro robot siga de manera constante una pared de perímetro circular mientras que mantiene 2uds de distancia con la misma haciendo uso tanto de la odometría como del sonar.

Para esto lo primero es instalar nuestro robot y mapa en el simulador tal y como se muestra en la imagen.



Tras esto, lo que hicimos fue inicializar las distintas variables que nos harán falta más adelante:

```
%% Inicializamos variables para el control
D = 1;
i = 0;
msg_sonar0 = receive (sonar0);
Kd = 1.8;
Ko = 0.18;
dist = 0;
lastpos = odom.LatestMessage.Pose.Pose.Position
lastdist = dist;
lastdistav = 0;
```

Como se puede ver, en un principio la posición inicial y lastpos son iguales por lo que decidimos mover un poco el robot gracias a las siguientes líneas de código:

```
msg_vel.Linear.X = 1; % velocidad lineal hacia adelante
send(pub, msg_vel); % enviar el mensaje de velocidad al robot
pause(1); % esperar 1 segundo
msg_vel.Linear.X = 0; % velocidad lineal hacia adelante
send(pub, msg_vel);
```

Esto lo hacemos porque al calcular la distancia avanzada necesitaremos que esta no nos de cero y al ser la formula: $\text{distav} = \sqrt{(\text{lastpos}.X - \text{pos}.X)^2 + (\text{lastpos}.Y - \text{pos}.Y)^2}$

Si las dos posiciones fuesen iguales la resta de las mismas daría cero provocando que el cálculo del error nos diese valores inválidos esto se debe a que las fórmulas de las mismas son:

$$E_{ori} = \text{atan2}(\text{abs}(\text{dist} - \text{lastdist}), \text{distav})$$

$$E_{dist} = ((\text{dist} + 0.105)) * \cos(E_{ori}) - D;$$

Por último, simplemente quedaría calcular las consignas de ambas velocidades que siguiendo el guion de la practica quedaria de la siguiente manera:

```
consigna_vel_linear = 0.3;
```

```
    consigna_vel_ang = Kd*Edist + Ko*Eori;
```

Todo esto mencionado puede verse en el archivo p2_p_YO.m

A) Documente la influencia de las ganancias del controlador en el seguimiento de la trayectoria (valores grandes y valores pequeños). Realice varios experimentos.

Al poner ambas ganancias a un valor parecido a 2 nuestro robot tarda un rato largo en encontrar la pared a la distancia que se desea, pero una vez que lo encuentra es capaz de cumplir con su funcionamiento de manera correcta. Un ejemplo del Eori que suele dar y de la distancia avanzadas es el mostrado en la siguiente imagen:

```
distav =  
  
    0.0299  
  
Eori =  
  
    single  
  
    0.5890  
  
    0.7400
```

Cuando ambas ganancias tienen un valor grande como puede ser 10, nuestro robot es incapaz de encontrar la pared a la distancia deseada por lo que empieza a girar en bucle de manera infinita.

Cuando ambos valores son próximos al cero (0.08) el robot gira de manera extremadamente lenta por lo que para cuando encuentra la pared y este intenta avanzar resulta que no está orientado a donde debería por lo que tras un rato de girar en bucle y moviéndose de manera no deseada este acaba por chocarse con la pared. Esto se puede ver en la siguiente imagen.

```

pos =

ROS Point message with properties:

  MessageType: 'geometry_msgs/Point'
    X: 14.7684
    Y: 14.1570
    Z: 0

  Use showdetails to show the contents of the mess:

distav =

    0.0309

Eori =

single

    0.5739

    3.0600

-----

pos =

ROS Point message with properties:

  MessageType: 'geometry_msgs/Point'
    X: 14.7906
    Y: 14.1385
    Z: 0

  Use showdetails to show the contents of the mess:

distav =

    0.0288

```

En la fotografía podemos ver como a pesar de que sí que existe una distav distinta de cero, la posición es prácticamente la misma en ambas fases.

Por último, decidimos probar con $K_d = 1.8$; y $K_o = 0.18$; . Con estos valores vimos que el robot es capaz de encontrar la pared bastante rápida y consigue seguirla de manera constante y manteniendo la distancia requerida. Es por esto que estos valores o cercanos a los mismos son los más eficientes para la ejecución de la práctica. Algunos valores que nos dará la prueba se muestran en la siguiente imagen:


```
-----  
  
pos =  
  
ROS Point message with properties:  
  
  MessageType: 'geometry_msgs/Point'  
    X: 12.0407  
    Y: 1.6679  
    Z: 0  
  
Use showdetails to show the contents of the mess  
  
distav =  
  
    0.0651  
  
Eori =  
  
  single  
  
    0.2982  
  
    0.8200
```

Cabe destacar que, en casi todas las fases, tanto eori como distav mantienen prácticamente los mismos valores.

B) Documente mediante simulaciones el efecto de anular Kd

Al anular Kd el robot es incapaz de detectar la pared a una distancia optima por lo que avanza hasta chocarse con esta. Al probar esto algunos de los valores que se nos muestran son los siguientes:

```
pos =
```

```
ROS Point message with properties:
```

```
  MessageType: 'geometry_msgs/Point'  
    X: 14.6314  
    Y: 17.3628  
    Z: 0
```

```
Use showdetails to show the contents of the mess:
```

```
distav =
```

```
0.0288
```

```
Eori =
```

```
single
```

```
0.6074
```

```
1.8000
```

B) Documente mediante simulaciones el efecto de anular Ko

Al anular Ko el robot sí que es capaz de encontrar la pared, aunque tarda mucho y necesita desplazarse demasiado sobre el mapa para hacerlo, pero a pesar de esto llega un momento en el que cumple con los requisitos de la práctica ya mencionados anteriormente.

C) 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.

Tras hacer varias pruebas, vemos que, si la velocidad lineal es demasiado alta, el robot avanza demasiado en el eje x por cada interacción del bucle que pasa. Esto acaba produciendo que este se choque con una pared antes de que el sensor sea capaz de detectarla y notificarlo.

Por otro lado, si la velocidad es baja, el sensor es capaz de analizar la distancia de la pared muchas más veces por cada metro que avanza, esto hace que sea capaz de analizar bien el entorno por lo que el robot funcionara correctamente.

D) 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.

A lo largo de esta práctica hemos podido ver que la realidad es muy distinta a la realidad ya que en esta sí que existen errores de medida que no podemos controlar, un ejemplo de esto es en el video que hemos añadido en la carpeta en el que se puede ver como el error de odometría provoca que el robot modifique la posición de las coordenadas (0,0).

Con el sonar estos errores también existen por lo que el robot no sería capaz de seguir la pared a 2 aunque si a una distancia similar. También existe la posibilidad de que en algún momento este error sea demasiado grande por lo que podría influir en el robot haciendo que este se pierda y no sea capaz de encontrarse de nuevo.

4. Conclusiones

La principal conclusión de esta práctica es la gran importancia que tienen los datos de salida de un dispositivo para el cálculo de sus mismas entradas. Esto lo hemos podido ver al programar ambos algoritmos de control ya que, en base a ciertos datos obtenidos en la salida de la interacción anterior, hemos podido calcular el error para orientar de nuevo a la máquina hacia la posición correcta.

Otra de las conclusiones que hemos sacado en claro de todo esto, es la importancia de los valores de las ganancias ya que son estos los que provocan la diferencia entre un buen funcionamiento o uno malo. Esto es algo que se nos ha mostrado en los distintos apartados que nos pedían modificar estos mismos para observar los distintos resultados que provocan.

5. Bibliografía

Para poder completar esta práctica nos hemos ayudado tanto de los archivos colgados por nuestros profesores como del mismo blog que matlab pone a disposición de los usuarios para que puedan solucionar algunos de sus errores.