

Sergio Sanz Cacho – 03201575K

Carlos Tejeda Martínez – 03148129G

TFA

RESOLUCIÓN DE LABERINTO

ÍNDICE

[INTRODUCCIÓN 2](#_Toc40988389)

[ESTRATEGIA EXPLORACIÓN 2](#_Toc40988390)

[MOVIMIENTO 5](#_Toc40988391)

[BUSQUEDA DE LA SALIDA 7](#_Toc40988392)

[FUNCIONES 8](#_Toc40988393)

# INTRODUCCIÓN

# ESTRATEGIA EXPLORACIÓN

A la hora de establecer un algoritmo de movimiento con el fin de recorrer todo el laberinto, nos basamos en el famoso y antiguo algoritmo usado para escapar de laberintos, la regla de la mano derecha.

REGLA DE LA MANO DERECHA: esta regla consiste en recorrer el laberinto con la mano apoyada en la pared derecha hasta llegar a su salida, dependiendo el tiempo de salida en la fortuna de iniciar a recorrer el laberinto en una posición ventajosa.

Esta regla ha sido implementada y en ambos laberintos el resultado ha sido positivo debido a que el robot no tarda gran tiempo ni repite trayectos a la hora de recorrer el laberinto. Esta implementación se explicará posteriormente en el apartado de MOVIMIENTO.

Gracias a esta manera de movimiento se consigue mapear el laberinto en alrededor de 4 minutos.

Para poder registrar y monitorear el avance en el mapeo del laberinto por parte del robot, hemos decidido implementar un mapa que registre las casillas por las que avanza el robot mediante una matriz de dimensiones igual de filas y columnas a los metros más 1 debido a que los arrays de Matlab trabajan como mínimo índice 1 y no 0.

Por ejemplo, en este caso para el laberinto de dimensiones 6m de alto y 14m de ancho, la matriz implementada será de 7 filas por 15 columnas:

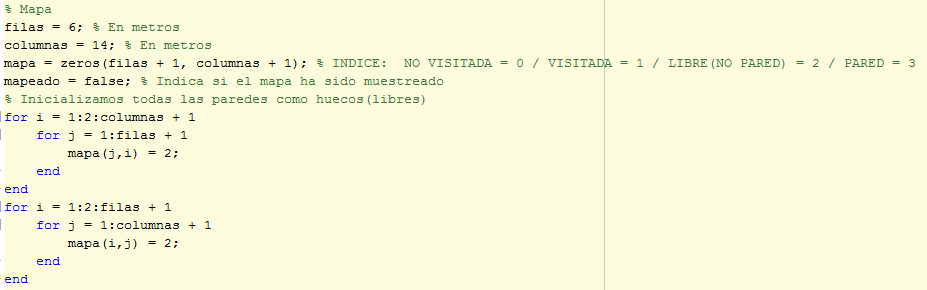


Ilustración 1-Implementación del mapa

Además, para poder registrar los diferente elementos de interés en el mapa, hemos definido un código relativo que será el siguiente para la interpretación del mapa:

* 0: Las posiciones de la matriz con un ‘0’ indica que son centros de casillas que aún no han sido alcanzados por el robot
* 1: Las posiciones de la matriz con un ‘1’ indica que son centros de casillas que han sido alcanzados por el robot.
* 2: Las posiciones de la matriz con un ‘2’ indica que son caminos libres por donde el robot puede transitar.
* 3: Las posiciones de la matriz con un ‘3’ indica que hay una pared por lo que el robot no puede avanzar a través de esta.
* Esquinas: Las esquinas de las casillas en la creación del mapa se quedarán con el valor ‘2’ asignado, debido a creación de la cuadrícula para el mapa y que no varían de valor en el mapeado debido a que son puntos irrelevantes y que únicamente serán tratados en la representación del mapa por pantalla.

Antes de iniciarnos en la actividad del robot y el uso del mapa creamos una cuadricula correspondiente con las posiciones en la matriz diseñada para representar los posibles caminos y paredes, que irán variando del dato inicial como libre a un dato real tras el paso del robot, a excepción de las esquinas no se modificarán ya que no serán necesarias en los procedimientos. Esta asignación se puede comprobar en la parte inferior con los dos bucles for de la ilustración 1.

Una vez preparada la base donde vamos a trabajar, entramos en un bucle while el cuál se ejecutará hasta que todo el laberinto haya sido registrado en la matriz, que será el encargado de mapear y mover el robot en este mismo orden.

Entrando en una nueva iteración del bucle, recogemos la posición y la rotación del robot, transformando el dato de la rotación obtenido en radianes en grados mediante una función que además corrige las pequeñas desviaciones que puede tener la medición, aproximando estas al ángulo más cercano de los usados para la ejecución del programa (0º,90º,180º y -90º).

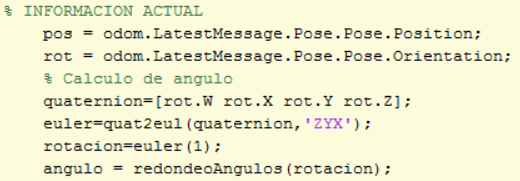


Ilustración 2-Obtención de la información principal de posición y orientación

Como último dato que recogemos a partir del robot antes del tratamiento de toda esta información, recabamos el tipo de casilla en el que nos encontramos mediante el uso del láser y la codificación especificada en la primera práctica.



Ilustración 3-Detección de tipo de casilla

Con este último dato obtenido del robot, nos iniciamos en el estudio de los valores para plasmarlos en el mapa mediante la función *actualizaMapa.* Esta función consiste en asignar en la matriz de manera correcta las paredes y accesos libres registrados desde la casilla visitada en función de la orientación del robot, ya que debido a esta varían los conceptos del lugar donde esta la pared. Antes de acabar y devolver el mapa actualizado, usamos una matriz auxiliar para representar el mapa por consola para comprobar el estado del mapeado usando símbolos representativos. Estos símbolos son los siguientes:

* ?: Las posiciones de la matriz con un ‘0’ se representarán por pantalla con el símbolo ‘?’ que significa que la casilla no ha sido visitada.
* V: Las posiciones de la matriz con un ‘1’ se representarán por pantalla con el símbolo ‘V’ que significa que la casilla ha sido visitada.
* L: Las posiciones de la matriz con un ‘2’ (excepto esquinas) se representarán por pantalla con el símbolo ‘L’ que significa que hay camino libre desde esa casilla.
* P: Las posiciones de la matriz con un ‘3’ se representarán por pantalla con el símbolo ‘P’ que significa que hay una pared.
* \_: Las esquinas al no ser un dato relevante, para mejorar la visualización de los restantes símbolos se muestran por pantalla como ‘\_’.

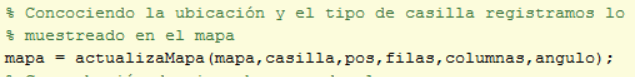


Ilustración -Actualización del mapa

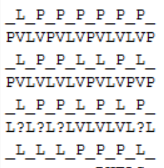


Ilustración 5-Salida por pantalla del estado del mapeado

Tras esto revisamos el estado del mapa para comprobar si con este último mapeamiento se ha finalizado la exploración de todas las casillas del laberinto mediante la función *horaDeSalir*, que registrará en la variable *mapeado* el estado.

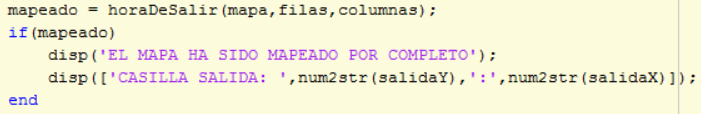


Ilustración 6-Comprobación de si el laberinto se ha mapeado

A continuación, se estudia el entorno más allá de la casilla en la que nos encontramos con el fin de tomar las futuras decisiones de movimiento y de comprobar si nos encontramos en la casilla de salida del laberinto, mediante la información volcada en la matriz de dos filas por 4 columnas a partir de la función *gps*. Esta matriz lo que indica en su primera fila con un ‘1’ en su posición es el de un posible camino para avanzar, mientras que su segunda fila con un ‘2’ en su posición indica una posible salida del laberinto; las columnas responden al orden respecto a la observación en el mapa de casilla superior, derecha, inferior e izquierda.



Ilustración 7-Estudio de opciones de movimiento y posible casilla de salida

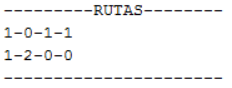


Ilustración 8-Salida por pantalla de la información alrededor de la casilla

Con esta información detectaremos si nos encontramos en la casilla de salida del laberinto mediante la salida de la función *luzAlFinalDelTunel*, que en caso de ser verdadera procedemos a almacenar la información de la casilla de salida.

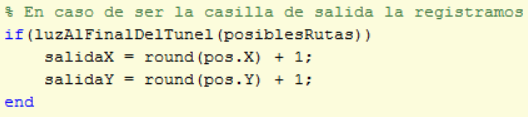


Ilustración 9-Detección y asignación de la casilla de salida

Como último paso antes de una nueva iteración ya solo resta el calculo del destino de la nueva casilla que visitará el robot mediante la función *siempreDerecha*, la cuál se encarga de respetar la lógica de movimiento siguiendo la pared derecha. Una vez conocido el centro de la nueva casilla, procedemos a movernos a esta para reiniciar todo este proceso de mapeamiento y añadir una nueva casilla y sus alrededores al mapa.

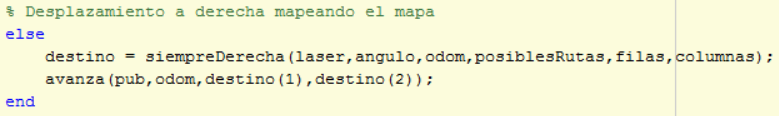


Ilustración 10-Movimiento a derechas del robot

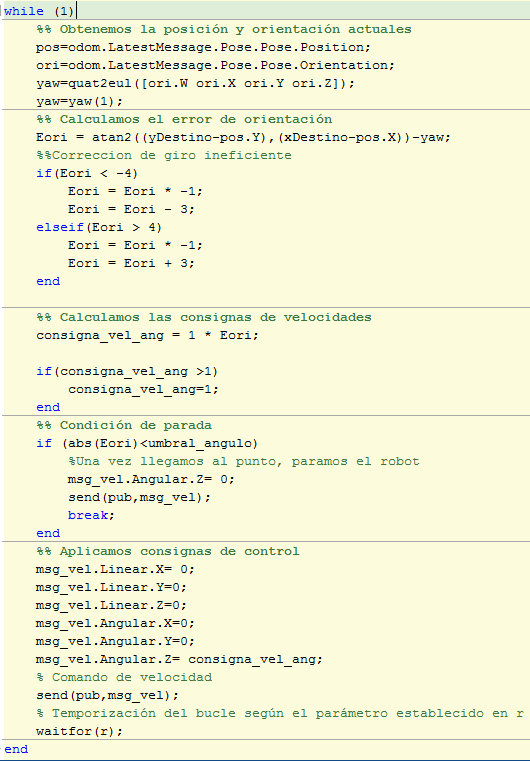
# MOVIMIENTO

Toda la lógica relacionada con el movimiento reside en la función *avanza*, la cuál consiste en mover el robot desde la casilla en la que se encuentra hacía el centro de la siguiente casilla anexa a esta, gracias a las coordenadas de esta nueva casilla facilitadas por argumento de esta función.

A diferencia que, en el resto del programa, las coordenadas con las que trabaja son las obtenidas directamente y entendidas por el robot y no las transformadas utilizadas en el mapa(desplazadas 1 unidad).

Este método se divide en dos bucles while, siendo el primero de ello encargado de la rotación y asegurándose que no se desvíe de la orientación requerida mediante un control P.

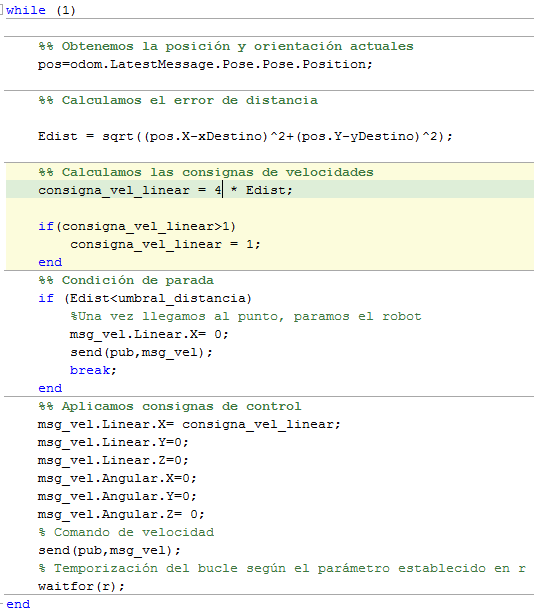
En este primer bucle también se dispone de un mecanismo que se asegura de que el robot nunca supere la velocidad máxima permitida de 1 rad/s, además de este método dispone de otro que se encarga de cambiar la dirección de giro en caso de que el error de orientación sea mas grande que el umbral establecido de +-4 radianes.



El segundo bucle se encarga de mover el robot a la siguiente casilla intentando llegar a la máxima velocidad posible mediante un control P.

Este segundo bucle es el encargado de llevar al robot en línea recta hasta las coordenadas destino, al igual que el mecanismo de control de la orientación también dispone de un sistema que se asegura de que no se supere la velocidad máxima de 1m/s.

Ambos sistemas el de orientación y el de avance tienen un margen de error inferior a 0.01 radianes en cuanto a la orientación con respecto el destino, e inferior a 0.05 m en cuanto a la posicion.



# BUSQUEDA DE LA SALIDA

Finalmente, cuando llegamos al punto en el que hemos logrado mapear todo el laberinto, pasamos a la última etapa de buscar la solución más rápida a la salida, conociendo en su totalidad el laberinto.

Cuando mapeamos la última casilla, no entramos en la opción de movimiento a derechas y entramos en la rama del final de la ejecución del programa mediante la indicación poniendo la variable *control* a false para no volver a ejecutar el bucle de mapeado.

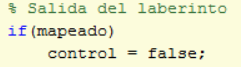


Ilustración 11-Marcar que esta es la última iteración

Una vez indicado el final de las iteraciones, se llama a la función recursiva *quieroSalir*, la cuál comprobará caminos hasta dar con el correcto para llegar a la casilla donde se encuentra la salida desde la casilla donde hemos acabado la actividad de explorar el laberinto. Esta función devuelve los centros de las casillas por los que debe pasar de manera secuencial el robot para llegar a la casilla final.

Ilustración 12-Búsqueda de ruta para la salida

Con esta secuencia, indicamos al robot que la recorra mediante el paso de estas coordenadas en un bucle que recorre el array devuelto.

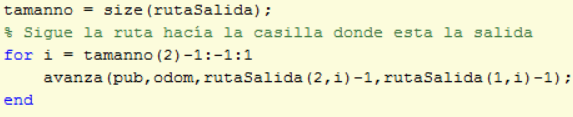


Ilustración 13-Avanzar el robot hacía la casilla de salida

Una vez llegamos a la casilla final, para realizar la acción de salida del laberinto para finalizar simulando el abandono del robot de este, hacemos un estudio de la casilla final análogo al realizado en la exploración del laberinto para encontrar la salida y recorrer el camino saliendo del laberinto.

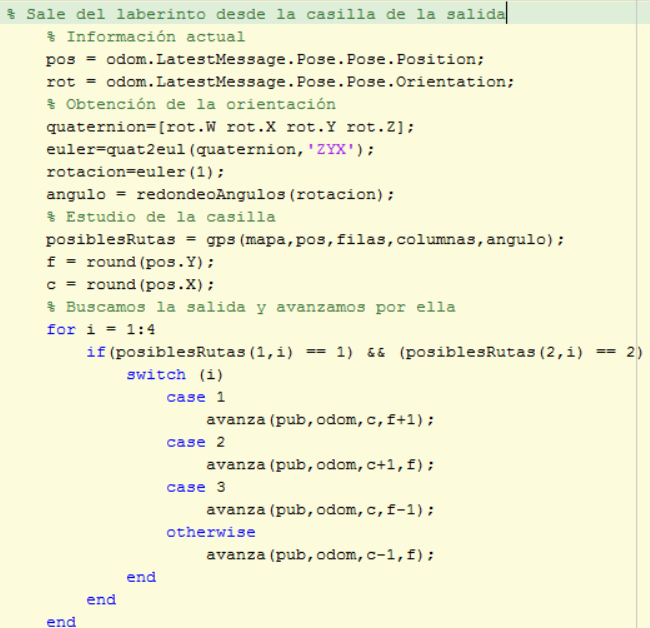


Ilustración 14-Simulación de salida del laberinto por parte del robot

# FUNCIONES

A continuación, vamos a realizar un breve resumen de las funciones creadas:

* *redondeoAngulos*: función encargada de convertir la orientación de radianes a grados, además de una vez hecha la conversión aproximar los valores a las cuatro orientaciones con las que trabajamos para sacar conclusiones a partir de los datos del robot. Estas orientaciones son: 0º, 90º, 180º y -90º.

Esta función recibe por parámetros la rotación del robot en radianes.

IMPLEMENTACIÓN:

%% FUNCIÓN PARA EL REDONDEO DEL ANGULO

function angulo = redondeoAngulos(rotacion)

% Angulos de aproximación

angulos = [0,90,180,-90];

% Conversión de radianes a angulos

angulo = round((rotacion \* 180)/pi);

for i = 1:4

if(i == 3)

if((abs(angulo) - angulos(i))<5 && (abs(angulo) - angulos(i)>-5))

angulo = angulos(i);

i = 4;

end

else

if((angulo - angulos(i))<5 && (angulo - angulos(i)>-5))

angulo = angulos(i);

i = 4;

end

end

end

* *detectorCasilla*: función encargada de obtener el tipo de casilla en la que nos encontramos, a partir de los datos de los láseres facilitados por las funciones *laser\_p1*, *laser\_p2*, *laser\_p3* y *laser\_p4*.

Esta función trabaja con los datos de la practica 1, cada función de las cuatro mencionadas antes se encarga de mapear una pared, p1 pared izquierda del robot, p2 la del frente, p3 la de la derecha y p4 la trasera.

Estas funciones trabajan con una región de rayos situada en los cuatro centros del robot y estas regiones se encargan de comprobar la existencia o no de pared.

Tras realizar la comprobación se devuelve un número que corresponde al tipo de casilla uno de los 15 tipos prestablecidos cuando se realizado la practica 1.

* *actualizarMapa*: función encargada de registrar en la matriz usada como mapa con los datos del entorno de la casilla.

En primer lugar, transformamos la posición donde nos encontramos a la relativa en el mapa (desplazamiento de 1), después marcamos en un array las paredes y espacios libres respecto a la orientación del robot para finalmente traducir en función de esta orientación a su posición correspondiente en la matriz.

Esta función recibe por parámetros la matriz del mapa, el tipo de casilla en la que estamos, posición actual, dimensiones de la matriz y el ángulo de la orientación.

IMPLEMENTACIÓN:

%% MÉTODO PARA LA ACTUALIZACIÓN DEL MAPA

function mapa = actualizaMapa(mapa,casilla,pos,filas,columnas,angulo)

% Obtenemos las coordenadas de la casilla actual

casillaX = round(pos.X) + 1;

casillaY = round(pos.Y) + 1;

% Marcamos como mapeada la casilla

mapa(casillaY,casillaX) = 1;

% Actualización de paredes

aux\_paredes = [2,2,2,2]; % INDICE: [FRENTRE ROBOT , DCHA ROBOT, DETRAS ROBOT, IZQ ROBOT]

% Frente robot

if (casilla == 1 || casilla == 5 || casilla == 6 || casilla == 7 || casilla == 11 || casilla == 13 || casilla == 14)

aux\_paredes(1) = 3;

end

% Derecha robot

if (casilla == 2 || casilla == 5 || casilla == 8 || casilla == 9 || casilla == 11 || casilla == 12 || casilla == 14)

aux\_paredes(2) = 3;

end

% Atrás robot

if (casilla == 3 || casilla == 6 || casilla == 8 || casilla == 10 || casilla == 11 || casilla == 12 || casilla == 13)

aux\_paredes(3) = 3;

end

% Izquierda robot

if (casilla == 4 || casilla == 7 || casilla == 9 || casilla == 10 || casilla == 12 || casilla == 13 || casilla == 14)

aux\_paredes(4) = 3;

end

if(angulo == 0)

% Casilla arriba

mapa(casillaY,casillaX+1) = aux\_paredes(1);

% Casilla derecha

mapa(casillaY+1,casillaX) = aux\_paredes(4);

% Casilla abajo

mapa(casillaY,casillaX-1) = aux\_paredes(3);

% Casilla izquierda

mapa(casillaY-1,casillaX) = aux\_paredes(2);

end

if(angulo == 90)

% Casilla arriba

mapa(casillaY,casillaX+1) = aux\_paredes(2);

% Casilla derecha

mapa(casillaY+1,casillaX) = aux\_paredes(1);

% Casilla abajo

mapa(casillaY,casillaX-1) = aux\_paredes(4);

% Casilla izquierda

mapa(casillaY-1,casillaX) = aux\_paredes(3);

end

if(angulo == 180)

% Casilla arriba

mapa(casillaY,casillaX+1) = aux\_paredes(3);

% Casilla derecha

mapa(casillaY+1,casillaX) = aux\_paredes(2);

% Casilla abajo

mapa(casillaY,casillaX-1) = aux\_paredes(1);

% Casilla izquierda

mapa(casillaY-1,casillaX) = aux\_paredes(4);

end

if(angulo == -90)

% Casilla arriba

mapa(casillaY,casillaX+1) = aux\_paredes(4);

% Casilla derecha

mapa(casillaY+1,casillaX) = aux\_paredes(3);

% Casilla abajo

mapa(casillaY,casillaX-1) = aux\_paredes(2);

% Casilla izquierda

mapa(casillaY-1,casillaX) = aux\_paredes(1);

end

% Representación del mapa

% Conversor

lista = zeros(filas+1,columnas+1);

conversorFilas = zeros(filas+1);

j = 1;

for i = filas+1:-1:1

conversorFilas(j) = i;

j = j + 1;

end

for i = 1:columnas+1

for j = 1:filas+1

lista(conversorFilas(j),i) = mapa(j,i);

end

end

% Pantalla

for i = 1:filas+1

concatenacion = "";

for j = 1:columnas+1

if(mod(i,2) == 1 && mod(j,2) == 1)

concatenacion = strcat(concatenacion,'\_');

else

switch lista(i,j)

case 0

concatenacion = strcat(concatenacion,'?');

case 1

concatenacion = strcat(concatenacion,'V');

case 2

concatenacion = strcat(concatenacion,'L');

case 3

concatenacion = strcat(concatenacion,'P');

end

end

end

disp (concatenacion);

end

* *horaDeSalir*: función encargada de comprobar si el laberinto ha sido recorrido en su totalidad.

Esto se consigue recorriendo en el mapa las posiciones que representan los centros de las casillas y en caso de que estén todas visitadas (casillas = ‘1’) se considera que la etapa de mapeamiento a finalizado devolviendo un true, en caso contrario devuelve false.

Esta función recibe por parámetros la matriz del mapa y las dimensiones de esta para poder recorrerlo.

IMPLEMENTACIÓN:

%% FUNCIÓN PARA COMPROBAR EN QUE MOMENTO PODEMOS SALIR DEL MAPA TRAS EL MAPEAMIENTO COMPLETO

function salir = horaDeSalir(mapa,filas,columnas)

% INDICE: 0 = no visitada // 1 = visitada // 2 = libre // 3 = pared

salir = true;

for i = 2:2:columnas + 1

for j = 2:2:filas + 1

if(mapa(j,i) == 1)

salir = true;

else

salir = false;

break;

end

end

if(~salir)

break;

end

end

* *gps*: función encargada de examinar posibles rutas a seguir y el contorno exterior a la casilla donde estamos para detectar posibles salidas del laberinto.

Esta función respecto a la orientación y la información del contorno de la casilla y las casillas anexas, resuelve en una matriz las posibles rutas que podemos seguir marcándolas con ‘1’ en la primera fila y marcando con ‘2’ en la segunda fila si la casilla anexa esta fuera del laberinto, para detectar la posible salida.

Esta función recibe por parámetros la matriz del mapa, la posición actual, las dimensiones del mapa y la orientación.

IMPLEMENTACIÓN:

function rutas = gps(mapa,pos,filas,columnas,angulo)

%% A PARTIR DEL TIPO DE CASILLAS INTERPRETAMOS LOS CAMINOS POSIBLES

% [RECTO / DERECHA / ATRAS / IZQUIERDA]

rutas = zeros(2,4);

% Obtenemos las coordenadas de la casilla actual

casillaX = round(pos.X) + 1;

casillaY = round(pos.Y) + 1;

% Estudio del mapa

% INDICE: NO HAY CAMINO = 0 / HAY CAMINO = 1 / POSIBLE SALIDA = 2

% INDICE: [FRENTRE ROBOT , DCHA ROBOT, DETRAS ROBOT, IZQ ROBOT]

% RUTAS(1,X): CAMINOS POSIBLES

% TURAS(2,X): POSIBLES SALIDAS

if(angulo == 0)%-----------------------------------------------------------

% Recto libre

if(mapa(casillaY,casillaX+1)== 2)

rutas(1,1) = 1;%Recto

end

if(casillaX < columnas)

if(mapa(casillaY,casillaX+2)== 0)

rutas(2,1) = 1;%Recto

end

else

rutas(2,1) = 2;

end

% Izquierda libre

if(mapa(casillaY+1,casillaX)== 2)

rutas(1,4) = 1;%Izq

end

if(casillaY < filas)

if(mapa(casillaY+2,casillaX)== 0)

rutas(2,4) = 1;%Izq

end

else

rutas(2,4) = 2;

end

% Atras libre

if(mapa(casillaY,casillaX-1)== 2)

rutas(1,3) = 1;%Atras

end

if(casillaX > 2)

if(mapa(casillaY,casillaX-2)== 0)

rutas(2,3) = 1;%Atras

end

else

rutas(2,3) = 2;

end

% Derecha libre

if(mapa(casillaY-1,casillaX)== 2)

rutas(1,2) = 1;%Dcha

end

if(casillaY > 2)

if(mapa(casillaY-2,casillaX)== 0)

rutas(2,2) = 1;%Dcha

end

else

rutas(2,2) = 2;

end

end

if(angulo == 90)%---------------------------------------------------------

% Derecha libre

if(mapa(casillaY,casillaX+1)== 2)

rutas(1,2) = 1;%Dcha

end

if(casillaX < columnas)

if(mapa(casillaY,casillaX+2)== 0)

rutas(2,2) = 1;%Dcha

end

else

rutas(2,2) = 2;

end

% Recto libre

if(mapa(casillaY+1,casillaX)== 2)

rutas(1,1) = 1;%Recto

end

if(casillaY < filas)

if(mapa(casillaY+2,casillaX)== 0)

rutas(2,1) = 1;%Recto

end

else

rutas(2,1) = 2;

end

% Izquierda libre

if(mapa(casillaY,casillaX-1)== 2)

rutas(1,4) = 1;%Izq

end

if(casillaX > 2)

if(mapa(casillaY,casillaX-2)== 0)

rutas(2,4) = 1;%Izq

end

else

rutas(2,4) = 2;

end

% Atras libre

if(mapa(casillaY-1,casillaX)== 2)

rutas(1,3) = 1;%Atras

end

if(casillaY > 2)

if(mapa(casillaY-2,casillaX)== 0)

rutas(2,3) = 1;%Atras

end

else

rutas(2,3) = 2;

end

end

if(angulo == 180)%---------------------------------------------------------

% Atras libre

if(mapa(casillaY,casillaX+1)== 2)

rutas(1,3) = 1;%Atras

end

if(casillaX < columnas)

if(mapa(casillaY,casillaX+2)== 0)

rutas(2,3) = 1;%Atras

end

else

rutas(2,3) = 2;

end

% Derecha libre

if(mapa(casillaY+1,casillaX)== 2)

rutas(1,2) = 1;%Dcha

end

if(casillaY < filas)

if(mapa(casillaY+2,casillaX)== 0)

rutas(2,2) = 1;%Dcha

end

else

rutas(2,2) = 2;

end

% Recto libre

if(mapa(casillaY,casillaX-1)== 2)

rutas(1,1) = 1;%Recto

end

if(casillaX > 2)

if(mapa(casillaY,casillaX-2)== 0)

rutas(2,1) = 1;%Recto

end

else

rutas(2,1) = 2;

end

% Izquierda libre

if(mapa(casillaY-1,casillaX)== 2)

rutas(1,4) = 1;%Izq

end

if(casillaY > 2)

if(mapa(casillaY-2,casillaX)== 0)

rutas(2,4) = 1;%Izq

end

else

rutas(2,4) = 2;

end

end

if(angulo == -90)%---------------------------------------------------------

% Izquierda libre

if(mapa(casillaY,casillaX+1)== 2)

rutas(1,4) = 1;%Izq

end

if(casillaX < columnas)

if(mapa(casillaY,casillaX+2)== 0)

rutas(2,4) = 1;%Izq

end

else

rutas(2,4) = 2;

end

% Atrás libre

if(mapa(casillaY+1,casillaX)== 2)

rutas(1,3) = 1;%Atras

end

if(casillaY < filas)

if(mapa(casillaY+2,casillaX)== 0)

rutas(2,3) = 1;%Atras

end

else

rutas(2,3) = 2;

end

% Derecha libre

if(mapa(casillaY,casillaX-1)== 2)

rutas(1,2) = 1;%Dcha

end

if(casillaX > 2)

if(mapa(casillaY,casillaX-2)== 0)

rutas(2,2) = 1;%Dcha

end

else

rutas(2,2) = 2;

end

% Recto libre

if(mapa(casillaY-1,casillaX)== 2)

rutas(1,1) = 1;%Recto

end

if(casillaY > 2)

if(mapa(casillaY-2,casillaX)== 0)

rutas(2,1) = 1;%Recto

end

else

rutas(2,1) = 2;

end

end

disp('---------RUTAS--------');

disp([num2str(rutas(1,1)),'-',num2str(rutas(1,2)),'-',num2str(rutas(1,3)),'-',num2str(rutas(1,4))]);

disp([num2str(rutas(2,1)),'-',num2str(rutas(2,2)),'-',num2str(rutas(2,3)),'-',num2str(rutas(2,4))]);

disp('----------------------');

* *luzAlFinalDelTunel*: función encargada de comprobar si nos encontramos en la casilla de salida en función de la salida de la función *gps*.

Esta estudia las rutas posibles y a las casillas que se podrían avanzar que estarían fuera de la matriz, considerando que, si se puede llegar a una casilla exterior a la matriz, estamos en la salida, devolviendo true.

Este hecho se refleja cuando opciones(1,i) es igual a ‘1’ y opciones(2,i) es igual a ‘2’.

Esta función recibe por parámetros la matriz obtenida de la función *gps*.

IMPLEMENTACIÓN:

%% FUNCIÓN QUE COMPRUEBA SI ESTAMOS EN LA CASILLA DE SALIDA

function salida = luzAlFinalDelTunel(opciones)

% INDICE: CAMINO NULO = 0 / CAMINO POSIBLE = 1 / POSIBLE SALIDA = 2

salida = false;

for i = 1:4

if((opciones(2,i) == 2)&& (opciones(1,i) == 1))

salida = true;

end

end

* *quieroSalir*: función encargada de buscar una ruta entre la casilla de salida y la casilla en la que nos encontramos.

Esta función trabaja de manera recursiva mediante dos matrices, una de ellas guarda el posible camino y la otra casillas descartadas por donde pase el robot, buscando unir los dos puntos para poder desplazar al robot a la salida.

Esta función recibe por parámetros la matriz del mapa, sus dimensiones, la posición donde nos encontramos, la posición de la casilla de salida y las matrices antes mencionadas y sus índices.

IMPLEMENTACIÓN:

%% FUNCIÓN PARA ENCONTRAR RUTA DE SALIDA (RECURSIVA)

function rutaSalida = quieroSalir(mapa,filas,columnas,auxY,auxX,inicioY,inicioX,ruta,nulos,i,j)

rutas = zeros(4); % INDICE: CASIILLA SUP / CASILLA DCHA / CASILLA INF / CASILLA IZQ

% NO CAMINO = 0 / CAMINO = 1 / CAMINO NULO = 2

% Comprobamos rutas

r1j = true;

r1i = true;

if ((mapa(auxY+1,auxX) == 2) && (auxY+2 < filas+1))

for z = 1:j

if ((nulos(1,z) == auxY+2) && (nulos(2,z) == auxX))

r1j = false;

end

end

for z = 1:i

if ((ruta(1,z) == auxY+2) && (ruta(2,z) == auxX))

r1i = false;

end

end

if(r1j) && (r1i)

rutas(1) = 1;

end

end

r2j = true;

r2i = true;

if ((mapa(auxY,auxX+1) == 2) && (auxX+2 < columnas+1))

for z = 1:j

if ((nulos(1,z) == auxY) && (nulos(2,z) == auxX+2))

r2j = false;

end

end

for z = 1:i

if ((ruta(1,z) == auxY) && (ruta(2,z) == auxX+2))

r2i = false;

end

end

if(r2j) && (r2i)

rutas(2) = 1;

end

end

r3j = true;

r3i = true;

if ((mapa(auxY-1,auxX) == 2) && (auxY-2 > 0))

for z = 1:j

if ((nulos(1,z) == auxY-2) && (nulos(2,z) == auxX))

r3j = false;

end

end

for z = 1:i

if ((ruta(1,z) == auxY-2) && (ruta(2,z) == auxX))

r3i = false;

end

end

if(r3j) && (r3i)

rutas(3) = 1;

end

end

r4j = true;

r4i = true;

if ((mapa(auxY,auxX-1) == 2) && (auxX-2 > 0))

for z = 1:j

if ((nulos(1,z) == auxY) && (nulos(2,z) == auxX-2))

r4j = false;

end

end

for z = 1:i

if ((ruta(1,z) == auxY) && (ruta(2,z) == auxX-2))

r4i = false;

end

end

if(r4j) && (r4i)

rutas(4) = 1;

end

end

% Elegimos ruta

decision = 0;

for z = 1:4

if(rutas(z) == 1)

control = true;

while control

decision = randi(4);

if(rutas(decision) == 1)

control = false;

end

end

break;

end

end

%SALIDA--------------------------------------------------------------------

disp([num2str(auxY),':',num2str(auxX)]);

%--------------------------------------------------------------------------

switch decision

case 1

auxY = auxY + 2;

i = i + 1;

ruta(1,i) = auxY;

ruta(2,i) = auxX;

case 2

auxX = auxX + 2;

i = i + 1;

ruta(1,i) = auxY;

ruta(2,i) = auxX;

case 3

auxY = auxY - 2;

i = i + 1;

ruta(1,i) = auxY;

ruta(2,i) = auxX;

case 4

auxX = auxX - 2;

i = i + 1;

ruta(1,i) = auxY;

ruta(2,i) = auxX;

otherwise

j = j + 1;

nulos(1,j) = auxY;

nulos(2,j) = auxX;

i = i - 1;

auxY = ruta(1,i);

auxX = ruta(2,i);

end

% SALIDA-------------------------------------------------------------------

disp (['BUSCANDO... i:',num2str(i),' j:',num2str(j)]);

disp([num2str(rutas(1)),'-',num2str(rutas(2)),'-',num2str(rutas(3)),'-',num2str(rutas(4))]);

disp([num2str(auxY),':',num2str(auxX)]);

%--------------------------------------------------------------------------

% Comprobamos si hemos acabado ruta

if((inicioY == auxY) && (inicioX == auxX))

rutaAux = zeros(2,i);

for z = 1:i

rutaAux(1,z) = ruta(1,z);

rutaAux(2,z) = ruta(2,z);

end

rutaSalida = rutaAux;

% En caso contrario continuamos buscando

else

rutaSalida = quieroSalir(mapa,filas,columnas,auxY,auxX,inicioY,inicioX,ruta,nulos,i,j);

end

* siempreDerecha: función encargada de mantener la regla de movimiento prefijada de la mano derecha para explorar el mapa.

Esta función recibe por parámetros el suscriber del laser y el de la edometría, además recibe el ángulo actual del robot la cantidad de filas y columnas en las que esta dividido el mapa, y las posibles rutas que puede tomar.

Lo primero que hace es comprobar en que direcciones tiene paredes p1,p2,p3 pared izquierda, frente y pared derecha respectivamente.

A continuación, se procede a realizar una corrección, en caso de que el robot se encuentre en la casilla final, se evalúan todas las posibilidades de orientación para evitar que el robot se salga del laberinto y prosiga mapeando, esta corrección consiste es situar una pared “imaginaria” poniendo a 1 el identificador de pared correspondiente.

Tras esta comprobación se procede a calcular la siguiente casilla a la que debe ir el robot esto se hace mediante 2 pasos.

El primero consiste en calcular el giro que va a necesitar el robot, es decir cuanto tiene que girar para colocarse en la orientación que le dirige a la próxima casilla, esto se consigue mediante un algoritmo que evalúa, las paredes que tiene alrededor y en función de ellas va siempre siguiendo la pared de la derecha, de esta manera se consigue recorrer todo un laberinto indistintamente de su forma.

Una vez calculado este giro necesario se aplica a la rotación actual del robot y en función del angulo resultante, se calcula la próxima casilla destino del robot.

IMPLEMENTACIÓN:

%% SiempreDerecha

function casilla = siempreDerecha(laser,angulo,odom,posiblesRutas,filas,columnas)

pos=odom.LatestMessage.Pose.Pose.Position;

cuarto = 100;

format long

inc = laser.LatestMessage.AngleIncrement; %Incremento del angulo en cada rayo

dist = laser.LatestMessage.Ranges; %Array de distancias

angulo\_min = laser.LatestMessage.AngleMin;%Angulo minimo en rad

p1 = laser\_p1(angulo\_min,inc,cuarto,dist); %izq

p2 = laser\_p2(angulo\_min,inc,cuarto,dist); %centro

p3 = laser\_p3(angulo\_min,inc,cuarto,dist); %dcha

% DECODIFICADOR

if(luzAlFinalDelTunel(posiblesRutas))

if(angulo == 0)

if((round(pos.X) + 3) > columnas+1)

p2 = 1;

end

if((round(pos.Y) + 3) > filas+1)

p1 = 1;

end

if((round(pos.Y) - 3) < 0)

p3 = 1;

end

elseif(angulo == 90)

if((round(pos.X) + 3) > columnas+1)

p3 = 1;

end

if((round(pos.Y) + 3) > filas+1)

p2 = 1;

end

if((round(pos.X) - 3)<0)

p1 = 1;

end

elseif(angulo == 180)

if((round(pos.Y) + 3) > filas+1)

p3 = 1;

end

if((round(pos.X) - 3)<0)

p2 = 1;

end

if((round(pos.Y) - 3) < 0)

p1 = 1;

end

elseif(angulo == -90)

if((round(pos.X) + 3) > columnas+1)

p1 = 1;

end

if((round(pos.X) - 3)<0)

p3 = 1;

end

if((round(pos.Y) - 3) < 0)

p2 = 1;

end

end

end

if(~p1)&&(p2)&&(p3)

giro = 90;%+90

elseif(p1)&&(p2)&&(p3)

giro = 180;%+180

elseif(~p2)&&(p3)

giro = 0;%+0

elseif(~p3)

giro = -90;% -90

end

% Obtenemos las coordenadas de la casilla actua

casillaX = round(pos.X);

casillaY = round(pos.Y);

angulo = angulo + giro;

if(angulo>350)

angulo = 0;

elseif(angulo>250)

angulo = -90;

elseif(angulo<-350)

angulo = 0;

elseif(angulo<-250)

angulo = 90;

elseif(angulo<-170)

angulo = 180;

end

if(angulo == 0)

xDestino = casillaX + 2;

yDestino = casillaY;

elseif(angulo == 90)

xDestino = casillaX;

yDestino = casillaY+2;

elseif(angulo == 180)

xDestino = casillaX - 2;

yDestino = casillaY;

elseif(angulo == -90)

xDestino = casillaX;

yDestino = casillaY-2;

end

casilla = zeros(0,2);

casilla(1) = xDestino;

casilla(2) = yDestino;

* avanza: función encargada de mover al robot al centro de la siguiente casilla.

Explicada previamente en el apartado MOVIMIENTO.

La función avanza recibe el Publisher para enviar velocidades a los motores, el subscriber de la edometría y las coordenadas destino.

El mecanismo de avanza esta dividido en dos partes la primera es la orientación y la segunda es el avance.

La parte de la orientación trabaja mediante un controlador P, este se encarga de orientar el robot con el centro de la casilla destino(coordenadas destino), en esta parte también nos aseguramos de que la consigna de velocidad nunca supere 1 rad/s, también hemos implementado un mecanismo que se encarga de cambiar la dirección de giro en función de si el error de orientación es muy grande, así evitamos que el robot realice más giro del necesario. La orientación se consigue con un margen de error de 0.01 red.

La segunda parte es el avance, esta también controlada por un controlador P, además al igual que el anterior cuenta con un mecanismo para asegurase de que nunca supere 1m/s, el funcionamiento de esta parte es bastante sencillo le robot avanza en línea recta con la orientación que se le ha dado en la parte anterior de esta función, y se para cuando el error con respecto al destino es menor de 0.05 m.

IMPLEMENTACION:

function avanza(pub,odom,xDestino,yDestino)

%% DECLARACI?N DE MENSAGE

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);

%% Umbrales para condiciones de parada del robot

umbral\_distancia = 0.05;

umbral\_angulo = 0.01;

%% Bucles de control infinito

while (1)

%% Obtenemos la posici?n y orientaci?n 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 orientaci?n

Eori = atan2((yDestino-pos.Y),(xDestino-pos.X))-yaw;

%%Correccion de giro ineficiente

if(Eori < -4)

Eori = Eori \* -1;

Eori = Eori - 3;

elseif(Eori > 4)

Eori = Eori \* -1;

Eori = Eori + 3;

end

%% Calculamos las consignas de velocidades

consigna\_vel\_ang = 1 \* Eori;

if(consigna\_vel\_ang >1)

consigna\_vel\_ang=1;

end

%% Condici?n de parada

if (abs(Eori)<umbral\_angulo)

%Una vez llegamos al punto, paramos el robot

msg\_vel.Angular.Z= 0;

send(pub,msg\_vel);

break;

end

%% Aplicamos consignas de control

msg\_vel.Linear.X= 0;

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);

% Temporizaci?n del bucle seg?n el par?metro establecido en r

waitfor(r);

end

while (1)

%% Obtenemos la posici?n y orientaci?n actuales

pos=odom.LatestMessage.Pose.Pose.Position;

%% Calculamos el error de distancia

Edist = sqrt((pos.X-xDestino)^2+(pos.Y-yDestino)^2);

%% Calculamos las consignas de velocidades

consigna\_vel\_linear = 4 \* Edist;

if(consigna\_vel\_linear>1)

consigna\_vel\_linear = 1;

end

%% Condici?n de parada

if (Edist<umbral\_distancia)

%Una vez llegamos al punto, paramos el robot

msg\_vel.Linear.X= 0;

send(pub,msg\_vel);

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= 0;

% Comando de velocidad

send(pub,msg\_vel);

% Temporizaci?n del bucle seg?n el par?metro establecido en r

waitfor(r);

end