Trabajo Final de la Asignatura

Contenido

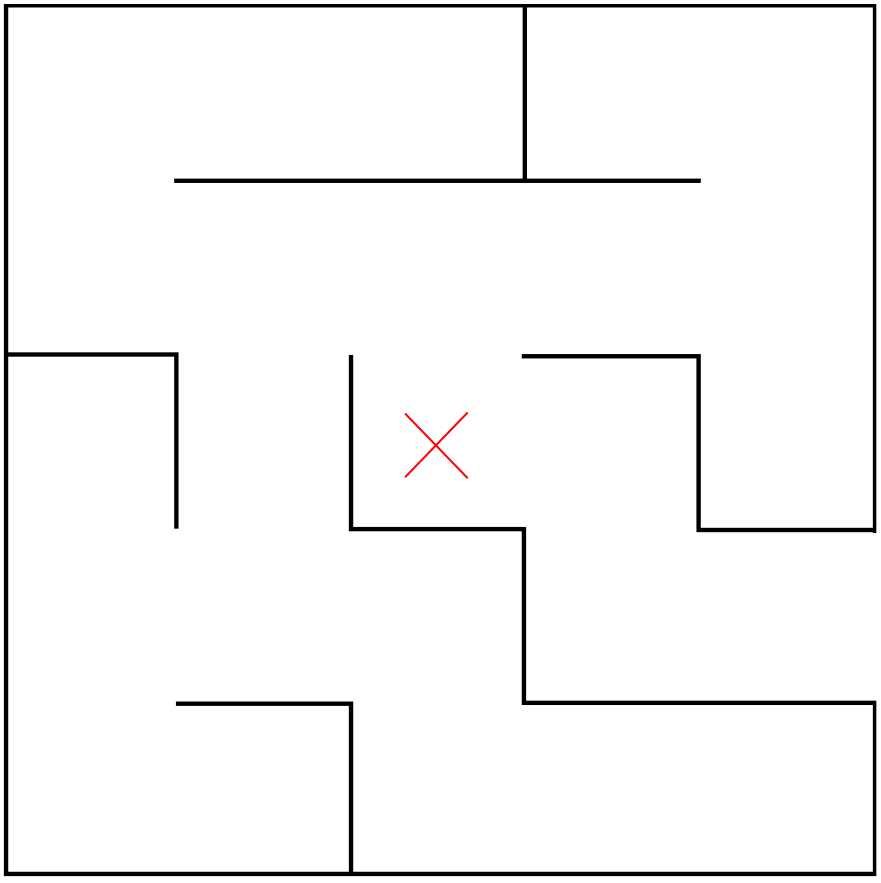
Grupo A2 Puesto 07
David Bachiller Vela
Pablo Bermejo

1. Resumen

Utilizando el robot Amigobot con el simulador, con el que se ha trabajado en las anteriores prácticas de la asignatura, se va a realizar la exploración de un entorno parcialmente estructurado. El mapa del entorno se desconoce a priori, pero se sabe que está estructurado en casillas de 4m x 4m y el tamaño total es de 20m x 30m y tiene una única salida, aun que el laberinto como tal es de 20m x 20m solo que hay que dejar espacio a la salida de ahí esos 10 metros de más, en total el robot recorrerá 27 celdas 25 correspondientes a laberinto y 2 a la salida, el robot ira recorriendo el mapa en base a un algoritmo de navegación basado en prioridad que consiste en dos patrones; El primero de ellos y más prioritario es el de ir a una celda que no se haya visitado, en segundo lugar la prioridad será en base lo que detecte que está libre al su alrededor, es decir el frente, izquierda, derecha o detrás, todo esto se sabrá gracias a la información recogida con los sonars, y el movimiento tanto linear y angular será gracias al sensor de odometría, esta prioridad se base en buscar la izquierda (lo que es más prioritario), si no el frente, si no la derecha y si no atrás, todo esto explicara con más detalle en la función **mapear,** que se podrá ver en la sección principal.

2. Sección Principal

Lo primero es saber como es el mapa donde se va a desarrollar todo el programa:



El robot se sitúa inicialmente en la posición (9,9), que será el punto de inicio y en base a eso comenzamos con la explicación detallada de las diferentes secciones de nuestro código.

2.1 Avanzar

Esta función la primera de nuestro codigo en MATLAB utiliza ROS (Robot Operating System) para controlar un robot y hacer que se desplace una distancia específica. A continuación, se resume su funcionamiento:

Primeramente, se cierra cualquier nodo ROS existente, luego se establece las variables de entorno de ROS para comunicarse con el maestro de ROS y especificar las direcciones IP de las máquinas que ejecutan MATLAB y Ubuntu y se Inicializa ROS con la dirección IP especificada.

Crea un suscriptor para recibir datos de odometría desde el tema "/robot0/odom".

Crea un publicador para enviar comandos de velocidad al tema "/robot0/cmd_vel" utilizando el tipo de mensaje "geometry_msgs/Twist".

Crea un objeto de mensaje del tipo "geometry_msgs/Twist" para contener los comandos de velocidad.

Establece el componente de velocidad lineal del mensaje (msg.Linear.X) en 0.8, lo que corresponde a una velocidad lineal deseada de 0.8 m/s.

Crea un objeto de frecuencia para controlar la frecuencia del bucle a 10 Hz. Espera 1 segundo para asegurarse de que se haya recibido un mensaje de odometría relacionado con el robot "robot0".

Inicializa la posición inicial utilizando los datos de posición del último mensaje de odometría.

Entra en un bucle de control infinito.

Obtiene la posición actual del último mensaje de odometría. Calcula la distancia euclidiana recorrida basada en la posición inicial y la posición actual. Muestra la distancia recorrida. Si la distancia recorrida supera la distancia deseada (distancia_recorrida), establece el componente de velocidad lineal del mensaje en 0 para detener el robot y sale del bucle de control.De lo contrario, envía el comando de velocidad para seguir moviendo el robot.

Espera la frecuencia especificada antes de iniciar la siguiente iteración del bucle de control.

2.2 Deteccion de paredes

Esta segunda sección de nuestro programa nos devuelve un booleano de las paredes que el robot, por lo que inicialmente y tras realizar las pertinentes conexiones. Se inicializan las variables paredes, **der**, **izq**, **front** y **back** para indicar la presencia de paredes en diferentes direcciones.

Se crean suscriptores para recibir los datos de los sensores de ultrasonido desde los diferentes temas de ROS:

```
sonarfront = rossubscriber('/robot0/sonar_2');
sonarback = rossubscriber('/robot0/sonar_6');
sonarback2 = rossubscriber('/robot0/sonar_7');
sonarder = rossubscriber('/robot0/sonar_5');
sonarizq = rossubscriber('/robot0/sonar_0');
```

Luego leemos el mensaje del sensor y extraemos la distancia a las paredes, definimos una variable llamada distm, que se refiere a la distancia máxima a la que el sensor va a detectar las paredes, esta la ponemos a una distancia de 2.5 metros.

Hacemos unas iteraciones comparando la distancia que extrae el sensor con la distancia máxima a la que detecta las paredes si es menor pues devolvemos paredes(x) = true;
Siendo Paredes(1) →Izquierda , Paredes(2) → Derecha , Paredes(3) → Frente y Paredes(4)→ Detrás.

2.3 Girar derecha

Esta función en MATLAB utiliza ROS (Robot Operating System) para controlar un robot y girarlo hacia la derecha en un ángulo específico que va a ser un giro de 90 grados. A continuación, se resume su funcionamiento:

Se crea un suscriptor para recibir datos de odometría desde el tema "/robot0/odom".

Se utiliza la función receive para recibir el último mensaje de odometría y asegurarse de que se haya recibido al menos un mensaje antes de continuar.

Se crea un publicador para enviar comandos de velocidad al tema "/robot0/cmd_vel" utilizando el tipo de mensaje "geometry_msgs/Twist".

Se crea un objeto de mensaje del tipo "geometry_msgs/Twist" para contener los comandos de velocidad.

Se obtiene la orientación inicial del robot desde el último mensaje de odometría y se calcula el ángulo en radianes (yaw_ini) utilizando la función quat2eul y rad2deg.

Se calcula el ángulo final de giro (yaw_fin) sumando un desplazamiento de -89 grados al ángulo inicial.

Se realizan ajustes para asegurarse de que el ángulo final esté dentro del rango adecuado (-180 a 180 grados).

Se redondea el ángulo final a un decimal para facilitar las comparaciones.

Se realizan comparaciones para determinar el valor final del ángulo (yaw_fin) según el caso específico (90 grados, -90 grados, 180 grados, -180 grados, o 0 grados).

Se define el margen de error del ángulo (errorAngulo).

Entra en un bucle de control infinito.

Dentro del bucle, se obtiene la orientación actual del robot desde el último mensaje de odometría y se calcula el ángulo actual (yaw_ac) utilizando las mismas conversiones que antes.

Se establecen los componentes de velocidad lineal y angular del mensaje según los valores específicos para girar hacia la derecha.

Si la diferencia absoluta entre el ángulo actual y el ángulo final es menor que el margen de error, se establecen los componentes de velocidad en cero para detener el robot y se envía el mensaje.

Se envía el mensaje de velocidad al publicador para continuar girando hacia la derecha.

El bucle se repite hasta que se alcanza el ángulo final deseado con suficiente precisión.

2.4 Girar Izquierda

Esta función en MATLAB utiliza ROS (Robot Operating System) para controlar un robot y girarlo en un ángulo de 90 grados hacia la izquierda, también se podía hacer invocando la función de girar derecha tres veces lo que nos ahorraba en líneas de código, pero el robot tenía que hacer mas recorrido, por lo que tardaba mucho mas así que le hemos diseñado una función propia. A continuación, se resume su funcionamiento:

e obtiene la orientación inicial del robot desde el último mensaje de odometría y se calcula el ángulo en radianes (yaw_ini) utilizando la función quat2eul y rad2deg.

Se suma el ángulo de giro deseado (giro) al ángulo inicial para obtener el ángulo final de giro (posFinal).

Se realizan ajustes para asegurarse de que el ángulo final esté dentro del rango adecuado (-180 a 180 grados).

Se convierte el ángulo final a radianes (yaw_fin).

Se redondea el ángulo final a un decimal para facilitar las comparaciones (arreglargiro).

Se define el margen de error del ángulo (errorAngulo).

Entra en un bucle de control infinito.

Dentro del bucle, se obtiene la orientación actual del robot desde el último mensaje de odometría y se calcula el ángulo actual (yaw_ac) utilizando las mismas conversiones que antes.

Se establecen los componentes de velocidad lineal y angular del mensaje según los valores específicos para girar hacia la izquierda.

Si la diferencia absoluta entre el ángulo actual y el ángulo final es menor que el margen de error, se establecen los componentes de velocidad en cero para detener el robot y se envía el mensaje.

Se envía el mensaje de velocidad al publicador para continuar girando hacia la izquierda.

El bucle se repite hasta que se alcanza el ángulo final deseado con suficiente precisión.

2.5 Avanzar atrás

La función avanzarAtras() realiza una secuencia de movimientos para que el robot avance hacia atrás.

Llamada a girarder(): Esta función se ejecuta una vez y hace que el robot gire hacia la derecha en un ángulo de 90 grados.

Llamada a girarder(): La función girarder() se ejecuta nuevamente, lo que hace que el robot gire nuevamente hacia la derecha en otros 90 grados, completando así una rotación total de 180 grados.

Llamada a avanzar(4.5): La función avanzar() se llama con un argumento de 4.5, lo que indica que el robot debe avanzar hacia adelante durante 4.5 unidades de distancia. El comportamiento exacto de la función avanzar() depende de su implementación específica.

Llamada a girarder(): Se ejecuta nuevamente la función girarder(), haciendo que el robot gire hacia la derecha en 90 grados.

Llamada a girarder(): La función girarder() se ejecuta por última vez, lo que hace que el robot gire hacia la derecha en otros 90 grados, completando una rotación total de 180 grados nuevamente.

En resumen, la función avanzarAtras() realiza una secuencia de movimientos en la que el robot gira dos veces hacia la derecha, avanza hacia atrás durante 4.5 unidades de distancia y luego gira dos veces más hacia la derecha.

2.6 Mapear

La función mapear() implementa un algoritmo de mapeo para un robot en un laberinto utilizando ROS (Robot Operating System) como plataforma de comunicación. El objetivo principal es explorar el laberinto y registrar las paredes y casillas visitadas para construir un mapa del entorno.

El algoritmo utiliza un enfoque basado en sensores para detectar las paredes que rodean al robot y determinar los posibles caminos disponibles. El robot se mueve en función de la prioridad establecida: primero se eligen las celdas no visitadas y luego se eligen las celdas libres en un orden específico (izquierda, frente, derecha, detrás). A medida que el robot se mueve, registra las coordenadas de las casillas visitadas en una matriz de mapa.

La función se ejecuta en un bucle hasta que se hayan visitado todas las casillas del laberinto. Durante cada iteración, se actualizan las coordenadas del robot, se registran las casillas visitadas en el mapa y se toman decisiones de movimiento en función de las condiciones del entorno.

En conclusión, la función mapear() proporciona un método sistemático para que un robot explore y mapee un laberinto utilizando información de sensores. Este enfoque permite al robot construir gradualmente un mapa del entorno y proporciona una base para realizar tareas más complejas, como la planificación de rutas o la localización en el laberinto.Inicialización de ROS: Se establecen las variables de entorno ROS_MASTER_URI y ROS_IP para establecer la conexión con ROS a través de la red.

Explicación de las principales funciones:

Declaración de suscriptores: Se crea un suscriptor para recibir la información de odometría del robot y otro suscriptor para recibir la información del sonar frontal.

Declaración de publicadores: Se crea un publicador para enviar los comandos de velocidad al robot.

Inicio de la función principal: Se inicializan las variables necesarias para el mapeo, como la matriz mapa que representa el mapa del laberinto, las coordenadas x e y del robot, y el contador de casillas visitadas.

Bucle principal: El bucle se repite hasta que se hayan visitado todas las casillas del laberinto (en este caso, 27 casillas). En cada iteración del bucle, se realizan las siguientes acciones:

Se utiliza la función detectarParedes() para obtener información sobre las paredes que rodean al robot.

Se cuenta el número de caminosLibres disponibles.

Se verifica si alguna celda libre ha sido visitada anteriormente. Para cada celda libre, se comprueba si las coordenadas x y y coinciden con alguna celda en la matriz mapa. Si coincide, se marca la celda como visitada y se incrementa el contador celdaVisitada.

Se elige una dirección de movimiento basada en la prioridad establecida. Si existen caminos no visitados, se elige el primer camino disponible en orden de prioridad (izquierda > frente > derecha > detrás) y se realiza el movimiento correspondiente.

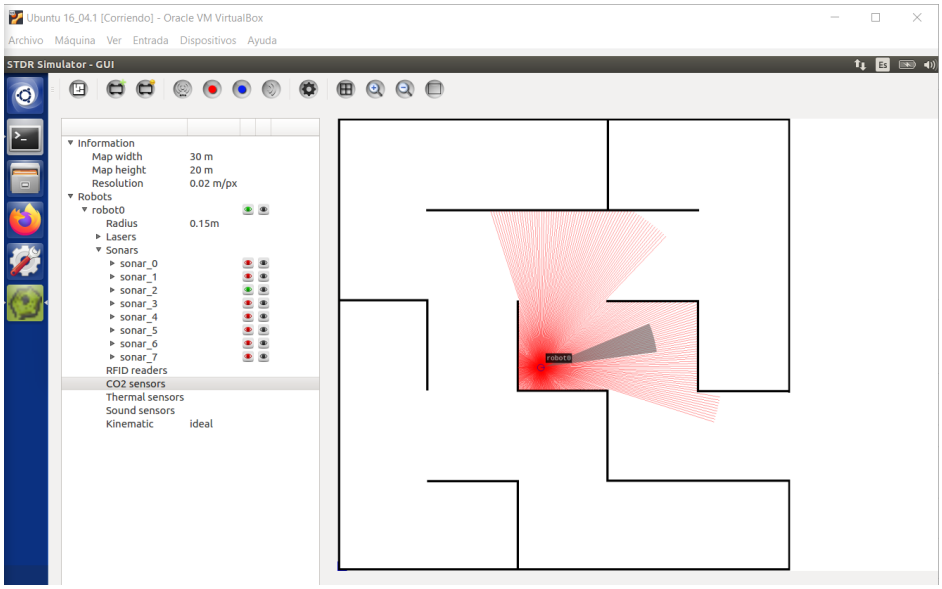
En caso de que todos los caminos hayan sido visitados, el robot retrocede hasta encontrar uno que este sin visitar, de esta manera logramos que salga de algunas celdas en las que se suelen quedar encerrado.

Se actualizan las coordenadas x e y según la dirección de movimiento elegida.

Se registra la información de la casilla visitada en la matriz mapa.

Se incrementa el contador de casillas visitadas.

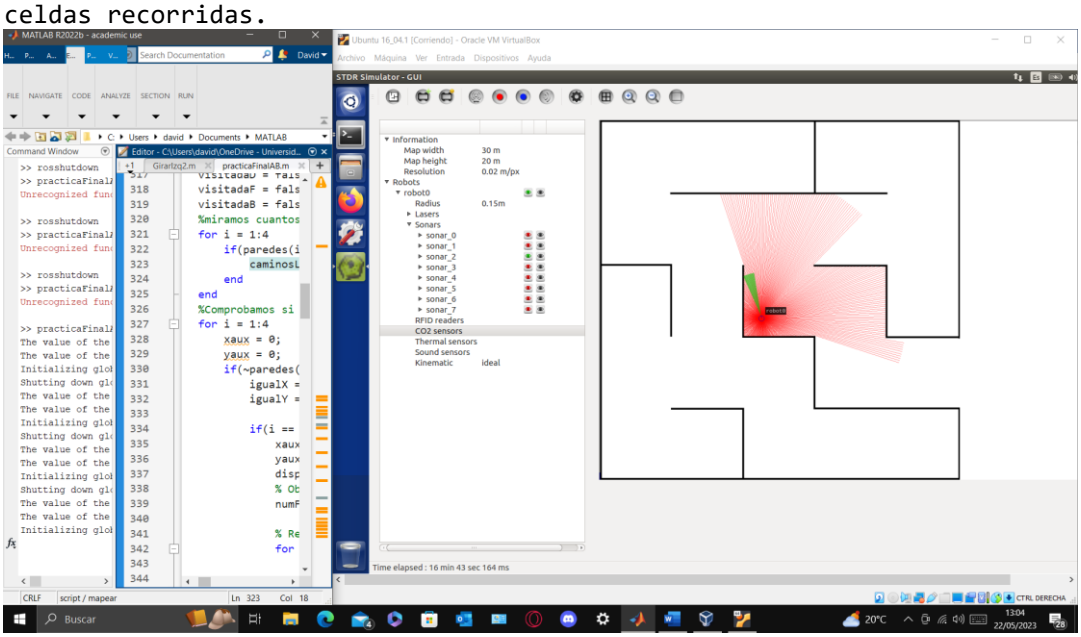
3. Resultados



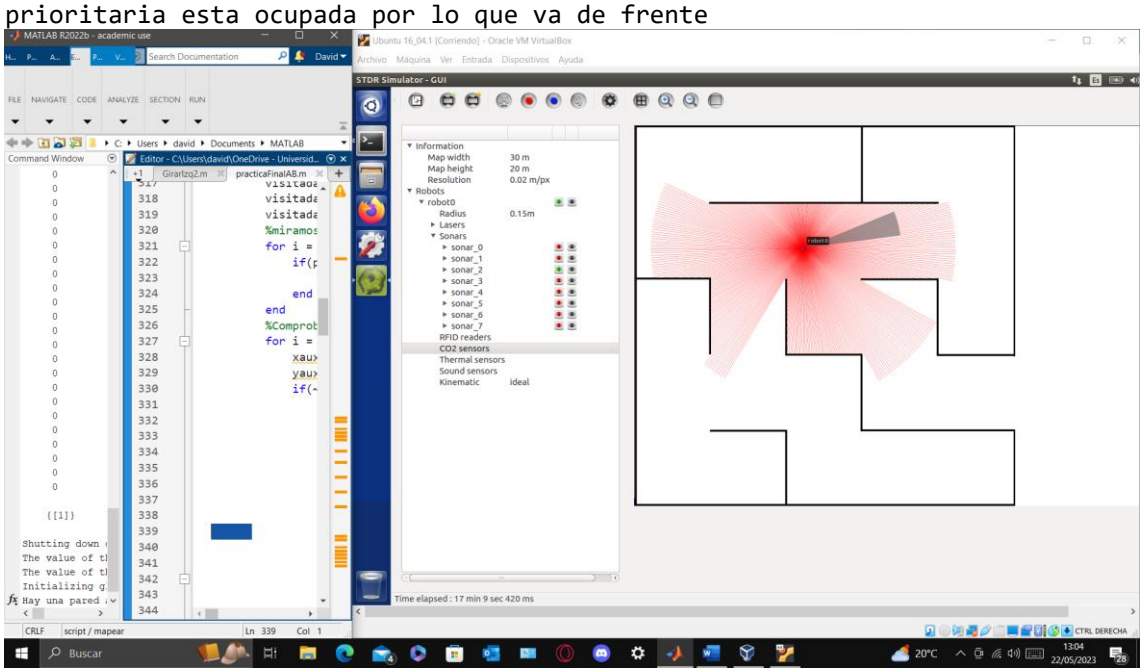

```
sonarfront = rossubscriber('/robot0/sonar_2');
sonarback = rossubscriber('/robot0/sonar_6');
sonarback2 = rossubscriber('/robot0/sonar_7');
sonarder = rossubscriber('/robot0/sonar_5');
sonarizq = rossubscriber('/robot0/sonar_0');
```

Dejo que se vea el sonar2 que es lo que va a marcar el frente, el robot parte de su posición inicial que es la (9,9) de ahí comienza su movimiento.

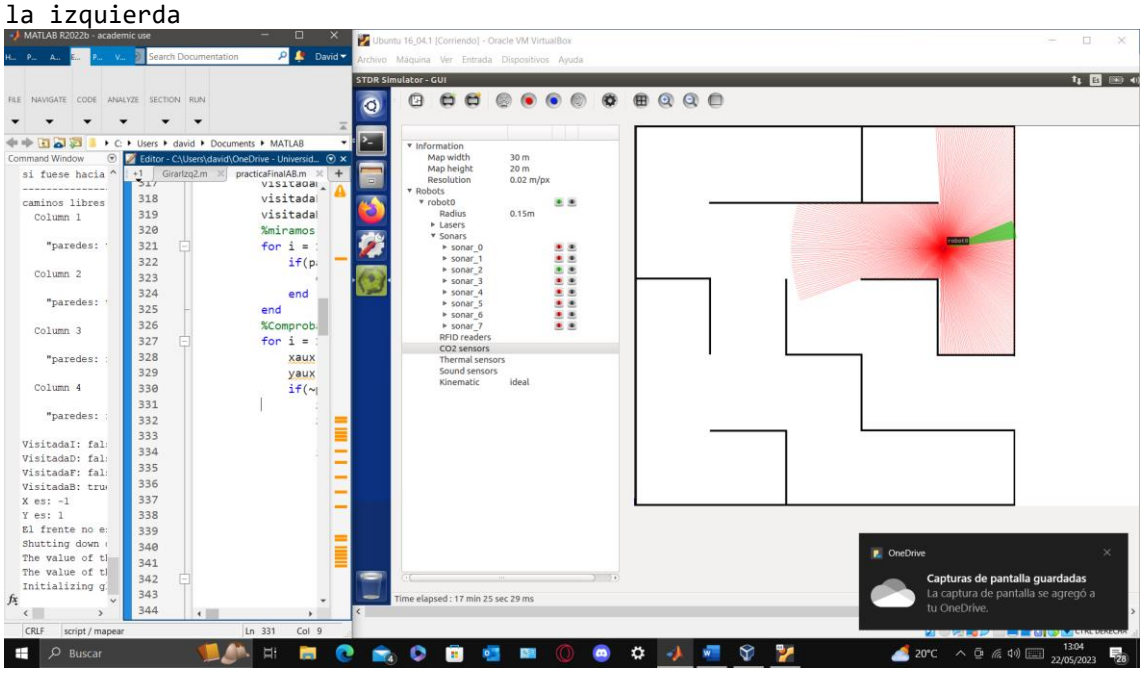
Primeramente detecta que hay celdas libres que es lo mas prioritario , por lo que ahora pasa a la segunda prioridad que es si la izquierda esta libre como lo esta avanza hacia ahí. Haciendo un giro de izquierdas avanzando y nuevamente un giro de derechas para posicionarse y sumándole uno al numero de celdas recorridas.

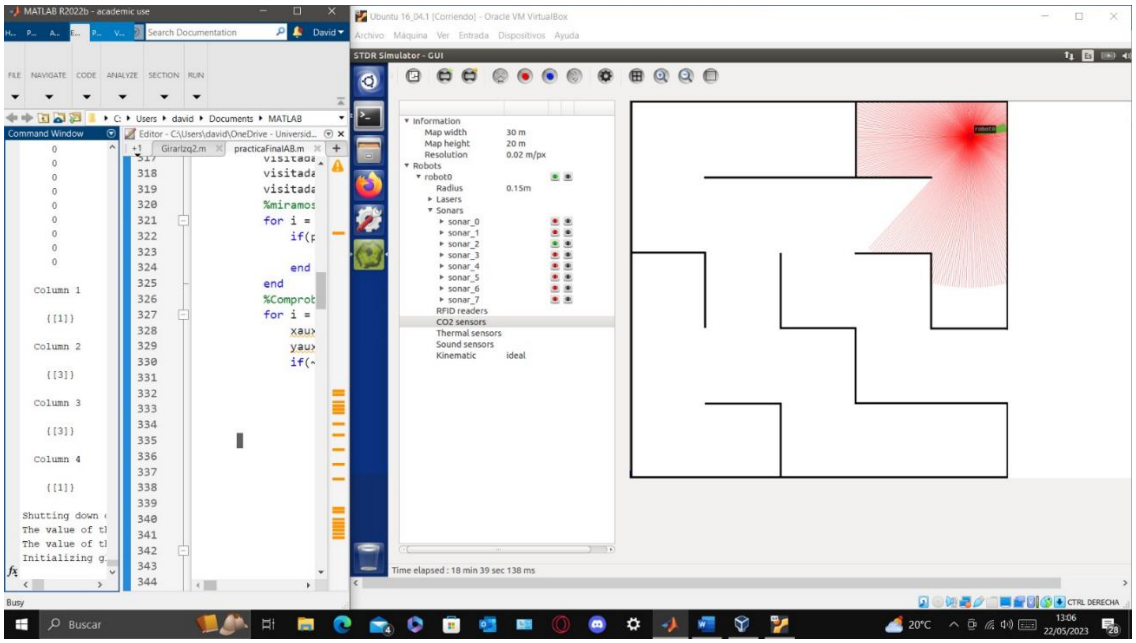


Luego analiza y ve casillas libres luego la izquierda que es la mas prioritaria esta ocupada por lo que va de frente

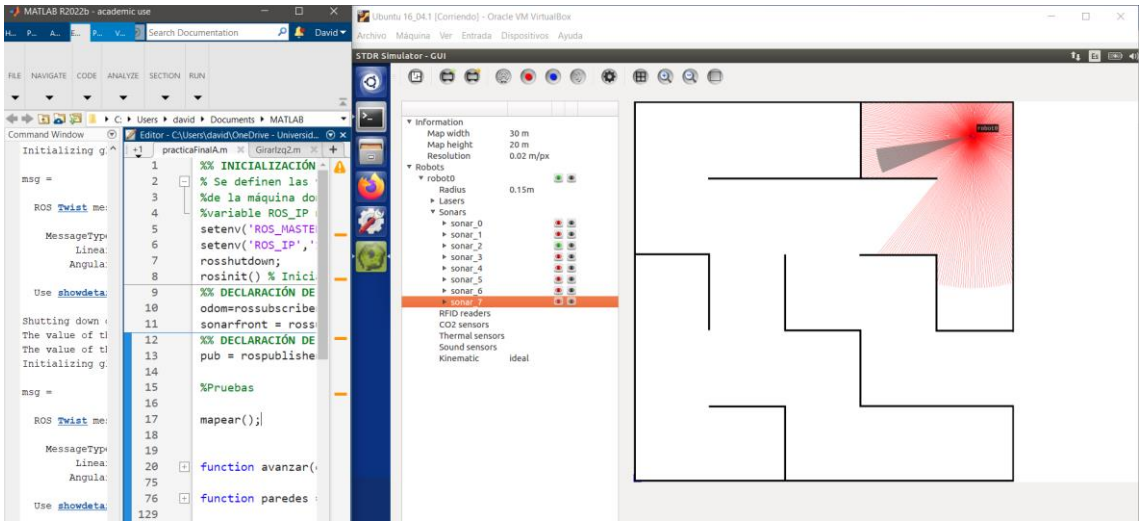


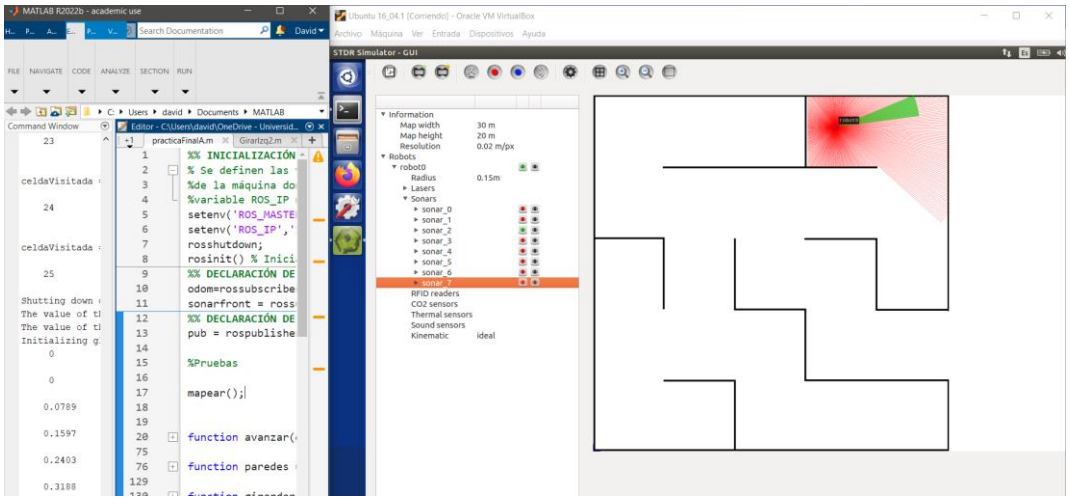
Llegados a este punto detecta casillas libres y la izquierda libre luego va a la izquierda



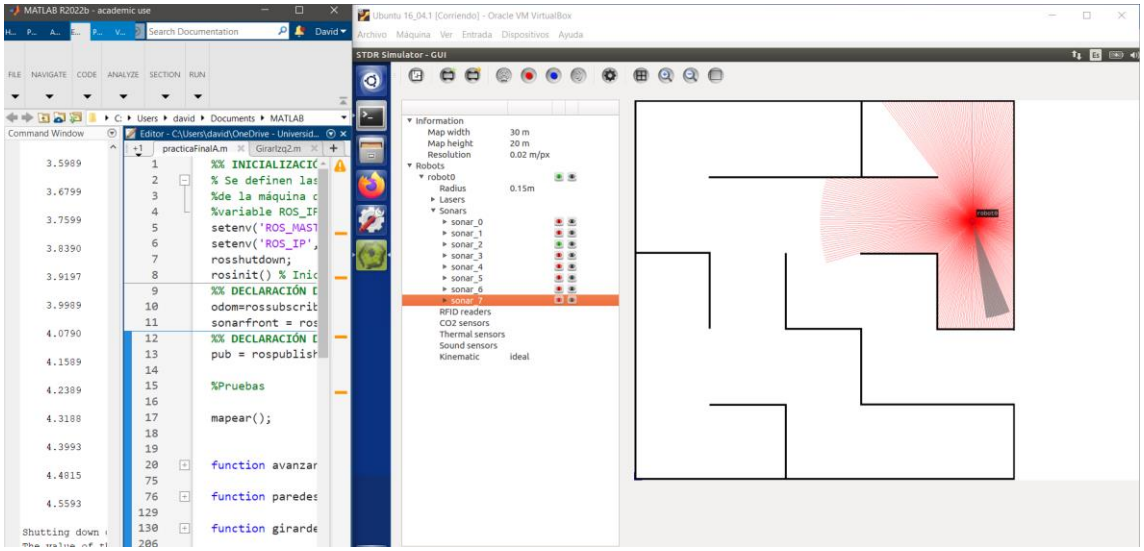


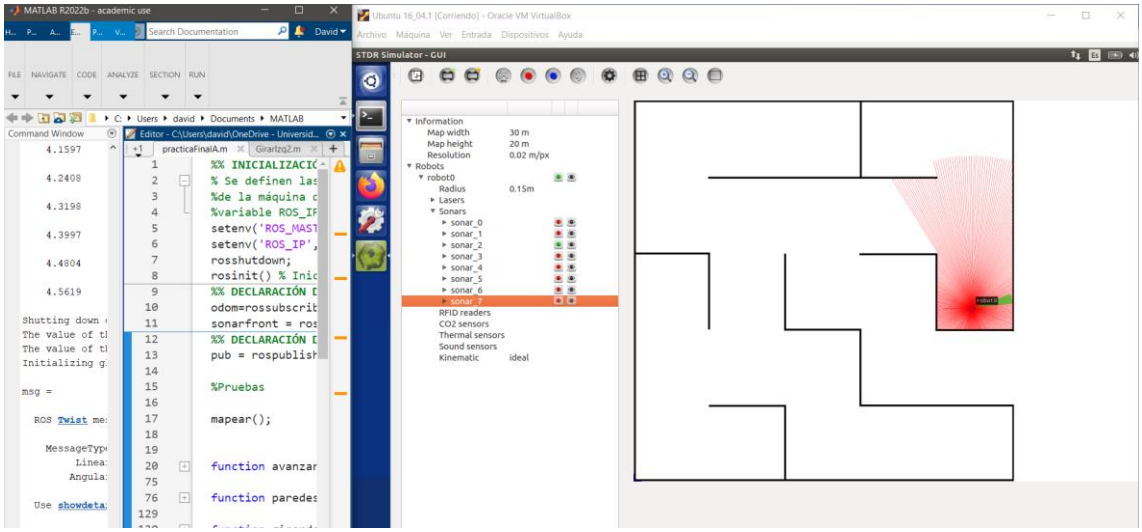
En este punto detecta la casilla de la izquierda libre y la de la derecha ocupada por lo que va hacia la de la izquierda con la función girar atrás haciendo la rotación y avanzando

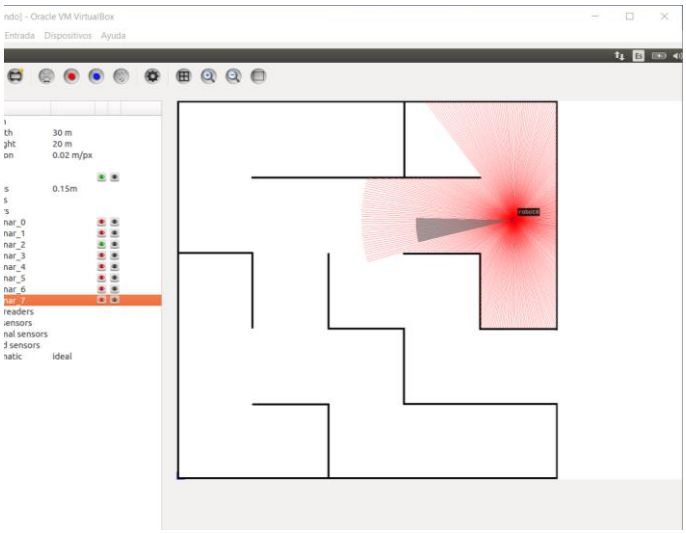




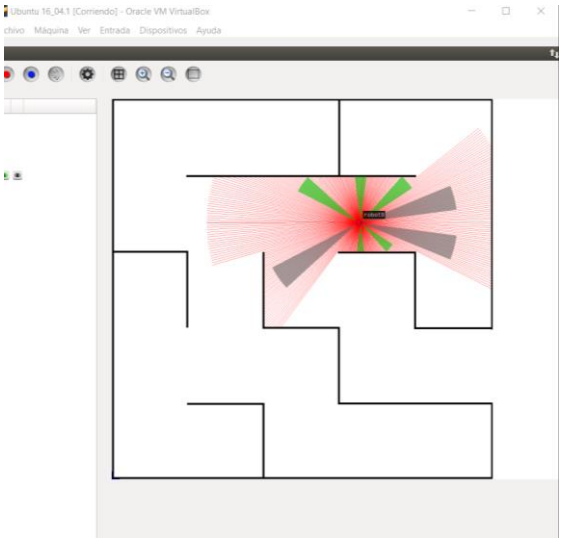
Luego vuelve y detecta la casilla libre luego va a ella

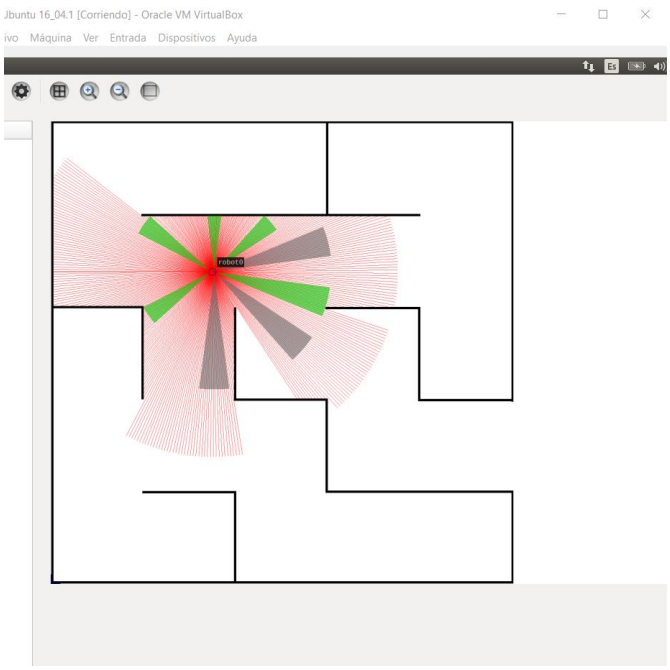




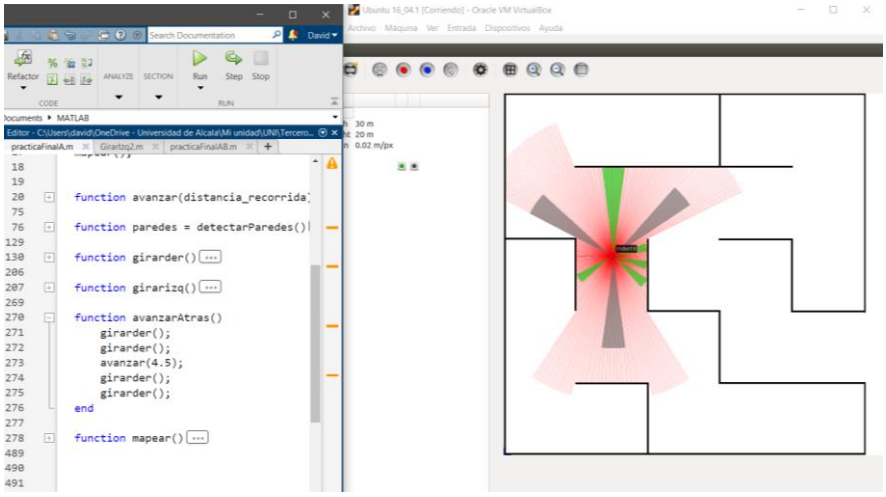


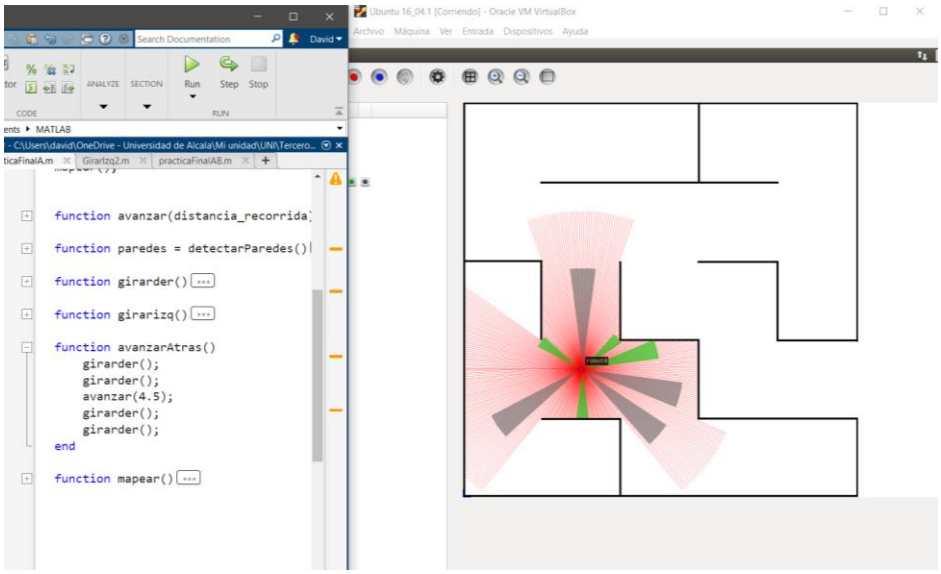
Va a seguir avanzando y detecta una casilla libre y una ocupada, luego sigue de frente hacia la casilla libre.



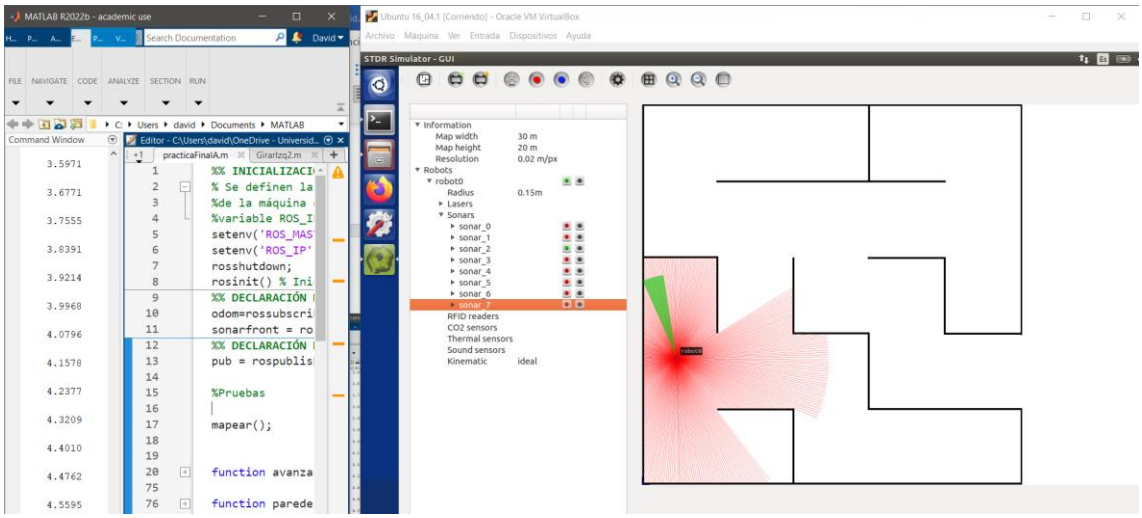


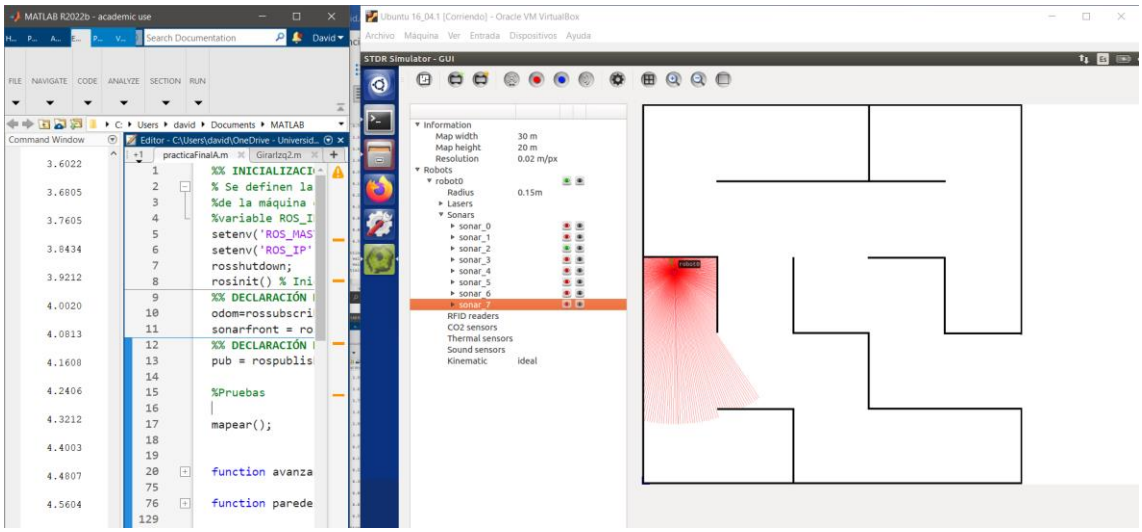
Detecta dos celdas libres luego va a la izquierda ya que la tiene libre

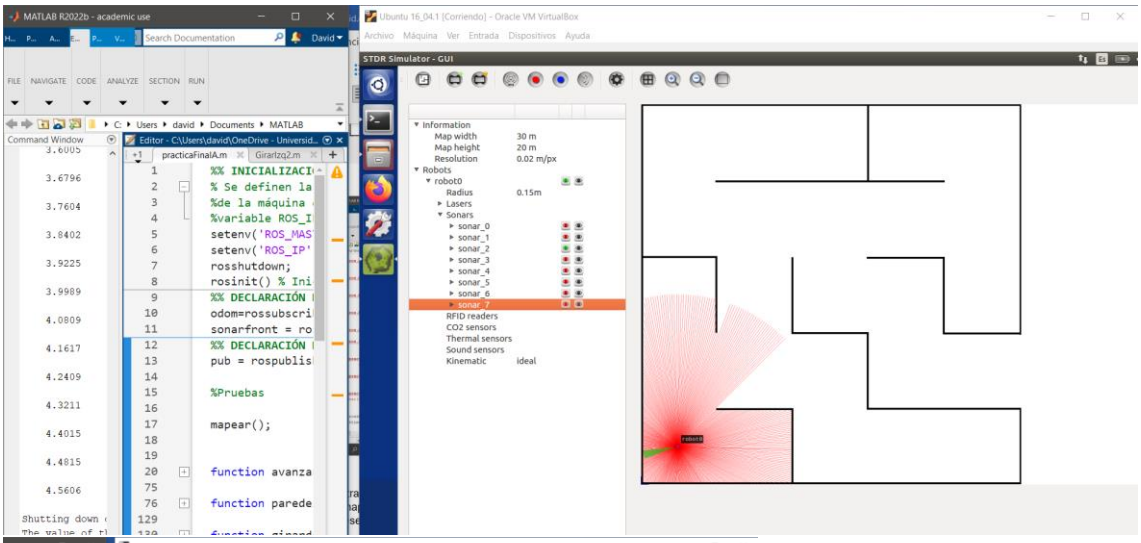


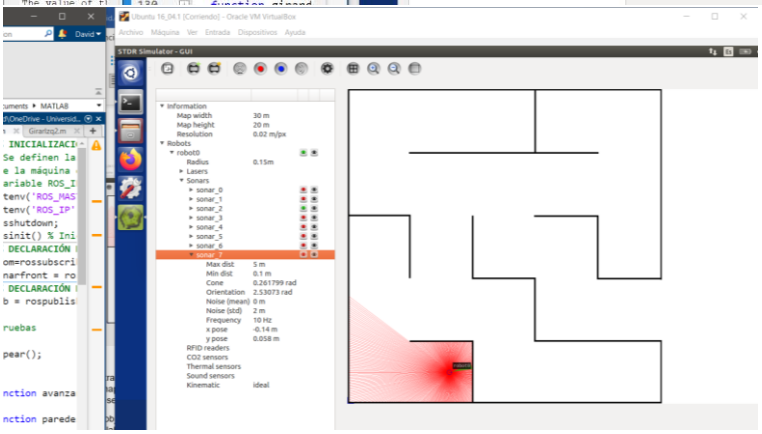


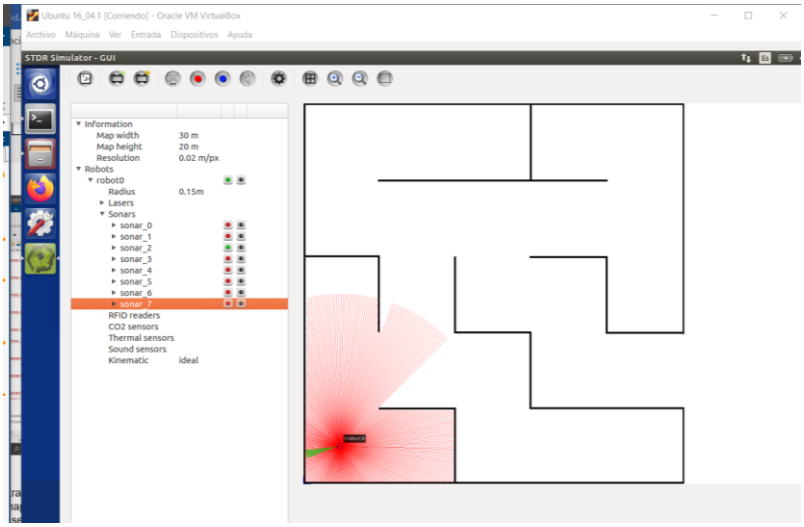
Luego detecta dos celdas libres a las que puede ir y la izquierda libre luego va a la izquierda

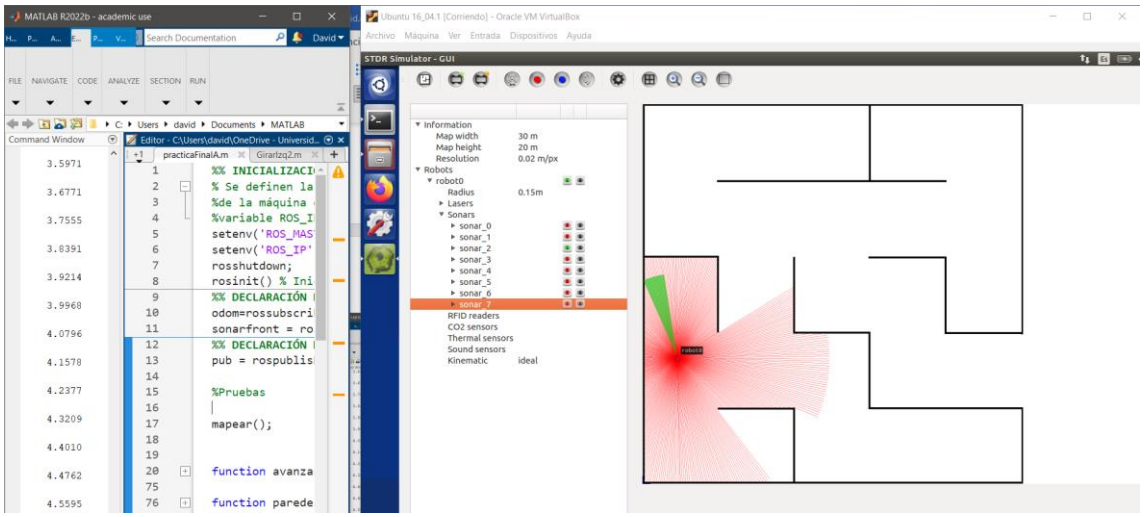


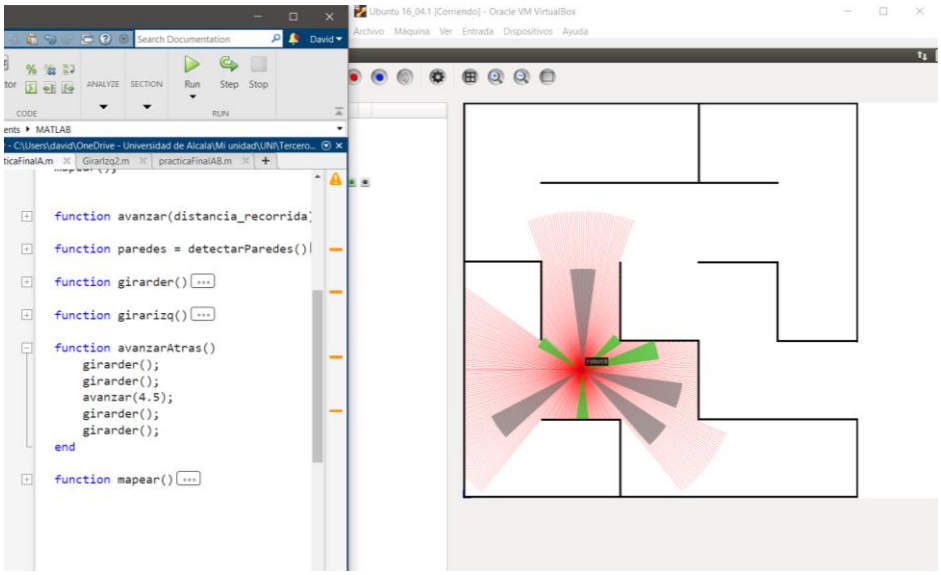


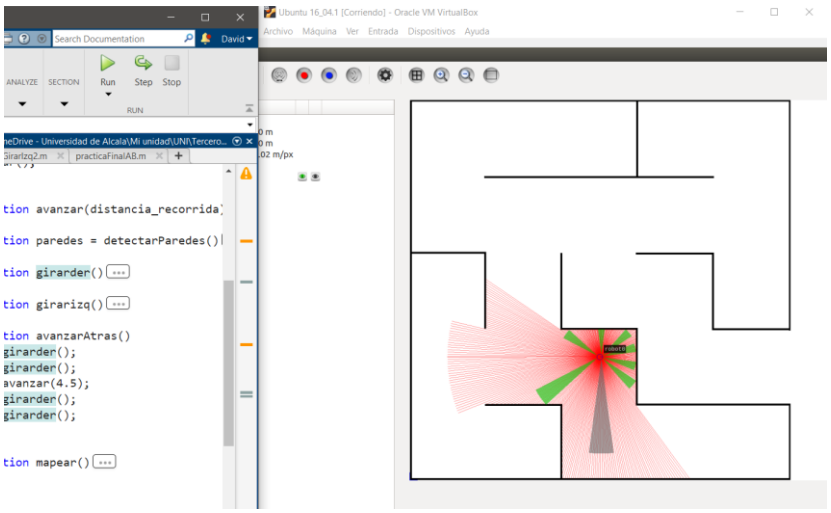

MATLAB R2022b - academic use
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
STDR Simulator - GUI
Information
Map width 30 m
Map height 20 m
Resolution 0.02 m/px
Robots
robot0
Radius 0.15m
Lasers
Sonars
RFID readers
CO2 sensors
Thermal sensors
Sound sensors
Kinematic ideal

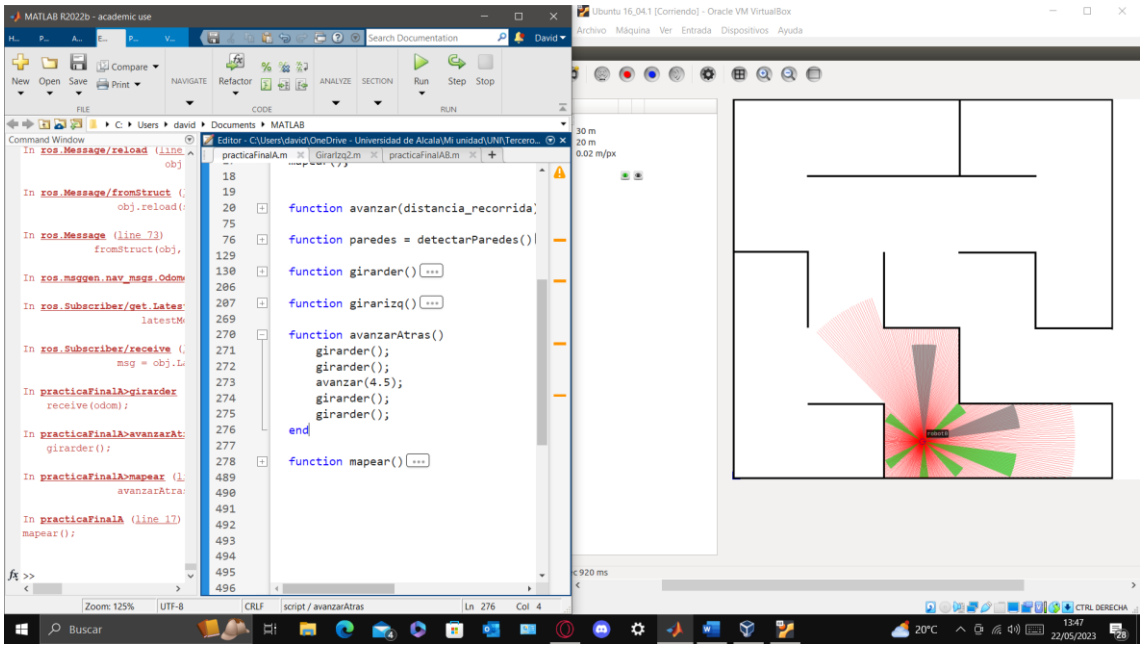


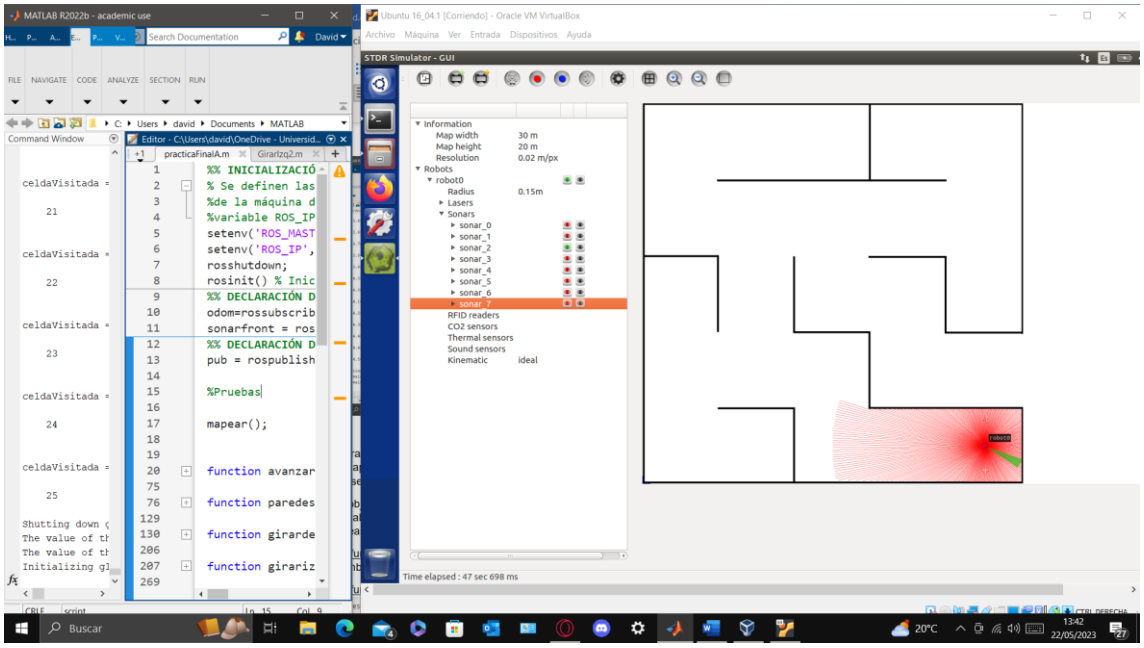

Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
STDR Simulator - GUI
Information
Map width 30 m
Map height 20 m
Resolution 0.02 m/px
Robots
robot0
Radius 0.15m
Lasers
Sonars
RFID readers
CO2 sensors
Thermal sensors
Sound sensors
Kinematic ideal

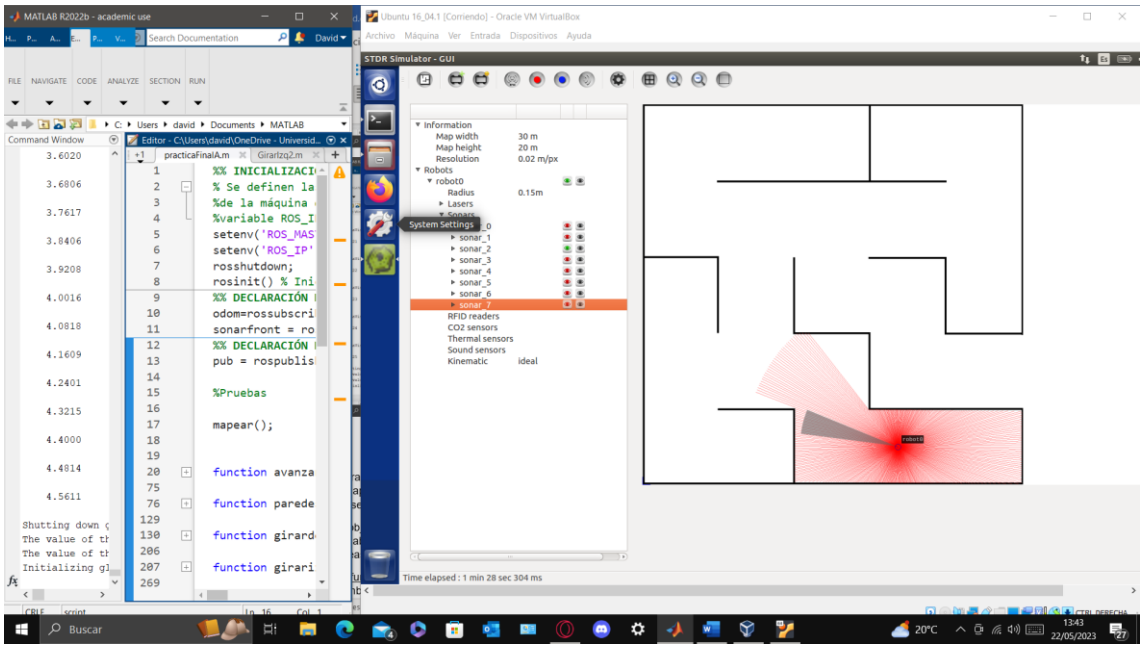

MATLAB R2022b - academic use
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
STDR Simulator - GUI
Information
Map width 30 m
Map height 20 m
Resolution 0.02 m/px
Robots
robot0
Radius 0.15m
Lasers
Sonars
RFID readers
CO2 sensors
Thermal sensors
Sound sensors
Kinematic ideal


function avanzar(distancia_recorrida)
function paredes = detectarParedes()
function girarder()
function girarizq()
function avanzarAtras()
girarder();
girarder();
avanzar(4.5);
girarder();
girarder();
end
function mapear()

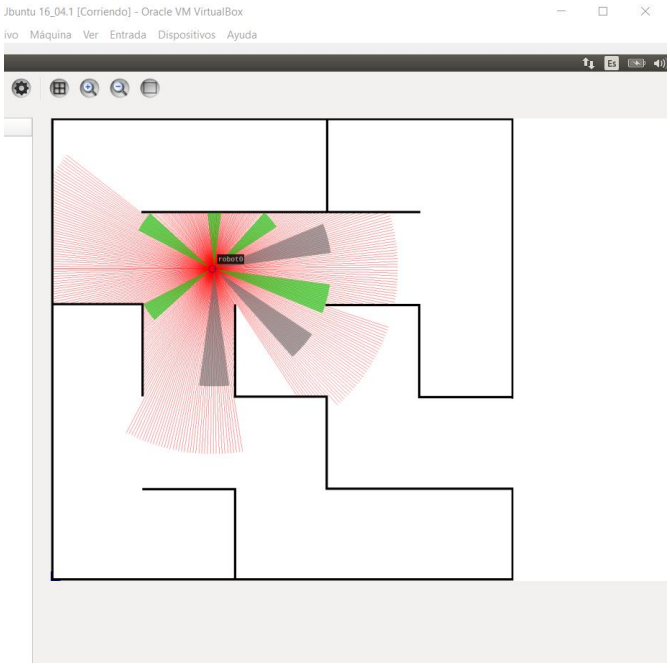

Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox


MATLAB R2022b - academic use
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
function avanzar(distancia_recorrida)
function paredes = detectarParedes()
function girarder()
function girarizq()
function avanzarAtras()
girarder();
girarder();
avanzar(4.5);
girarder();
girarder();
end
function mapear()

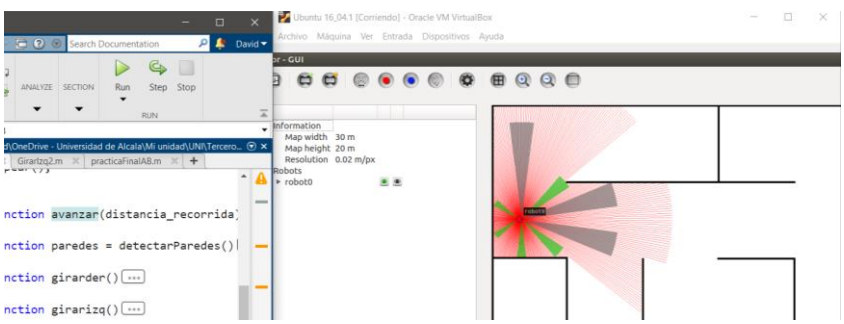

MATLAB R2022b - academic use
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
STDR Simulator - GUI
%% INICIALIZACIÓ
%Pruebas
mapear();

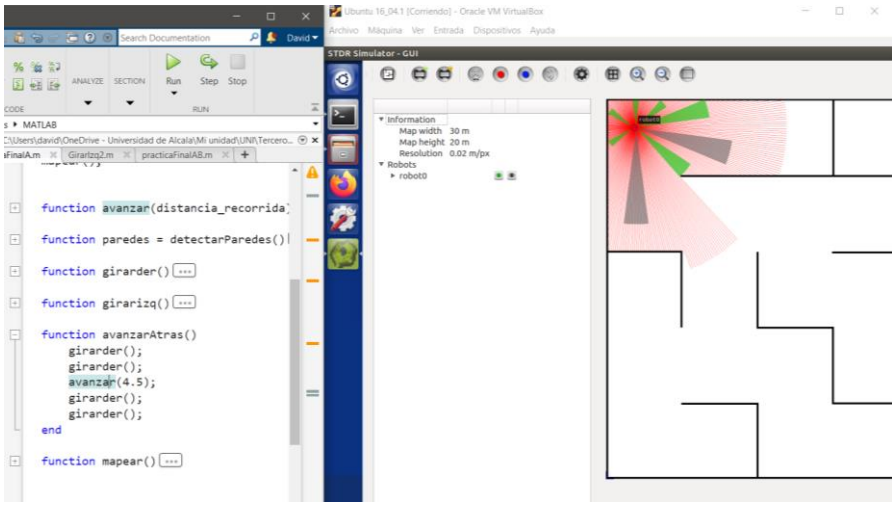


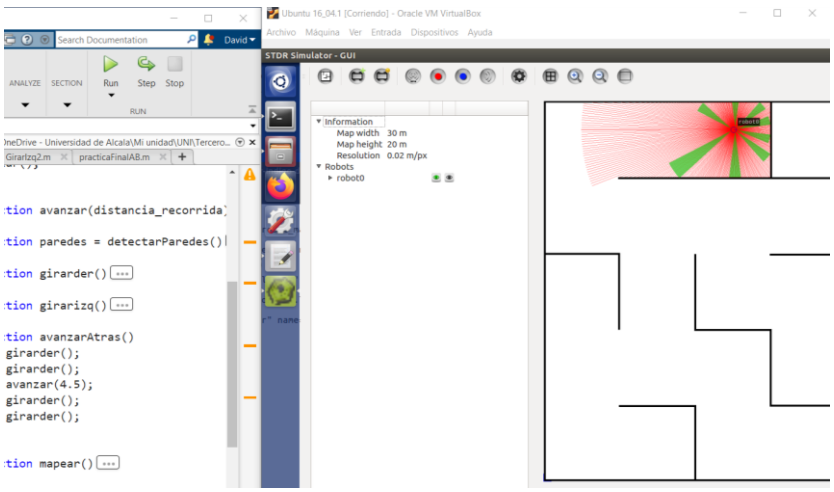
Ya va a hacer todo el recorrido de vuelta hasta:



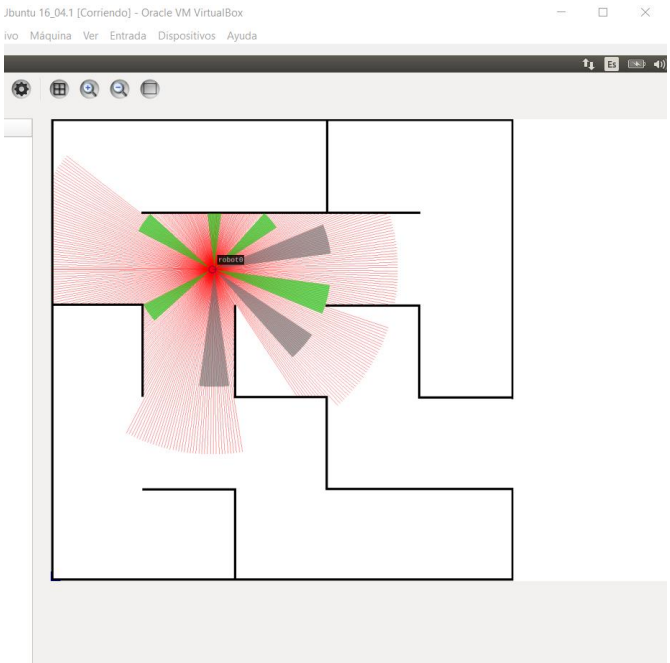
Luego detecta tres casillas dos de ellas están ocupadas y la de la izquierda esta libre luego va ahí.



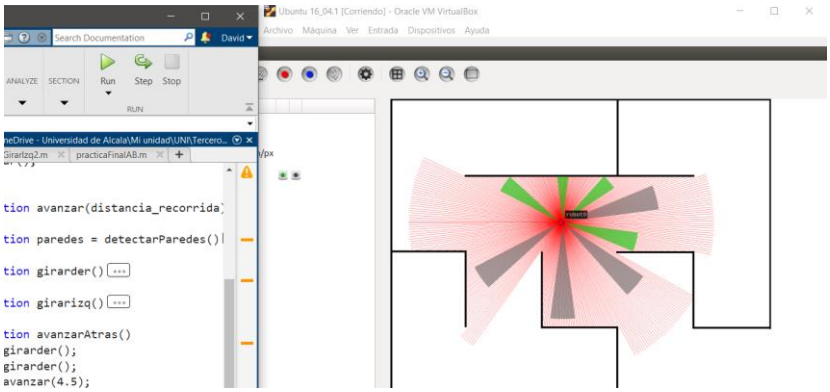


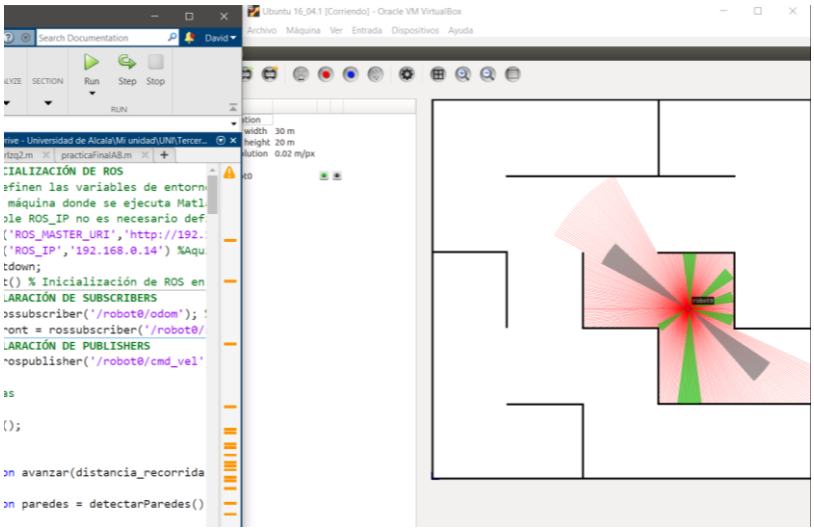


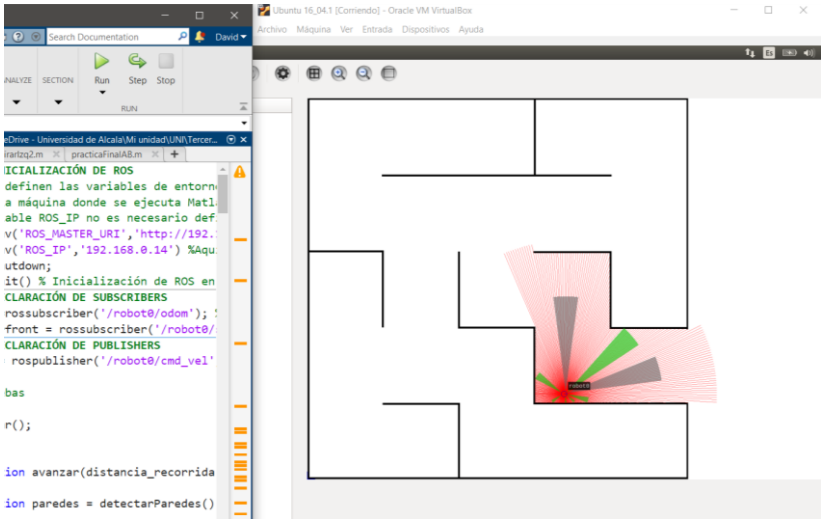
Ahora retrocede hasta volverse a encontrar

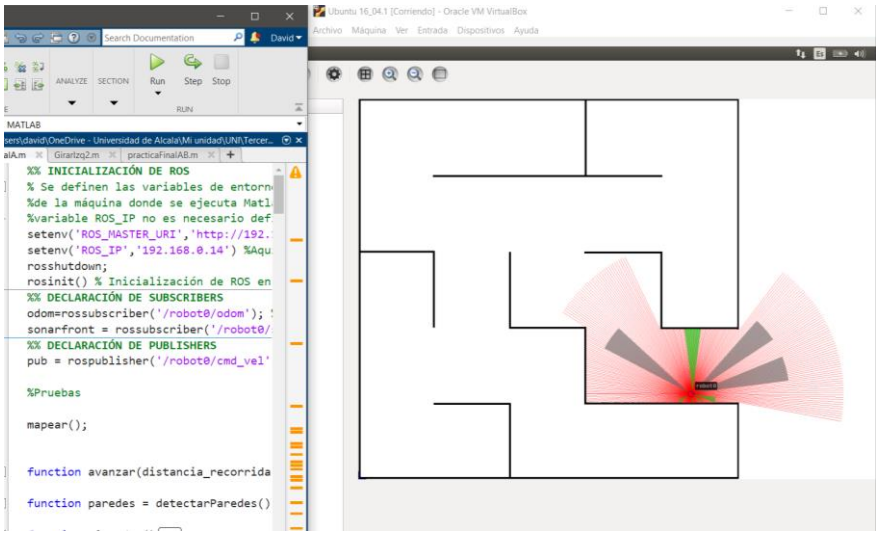


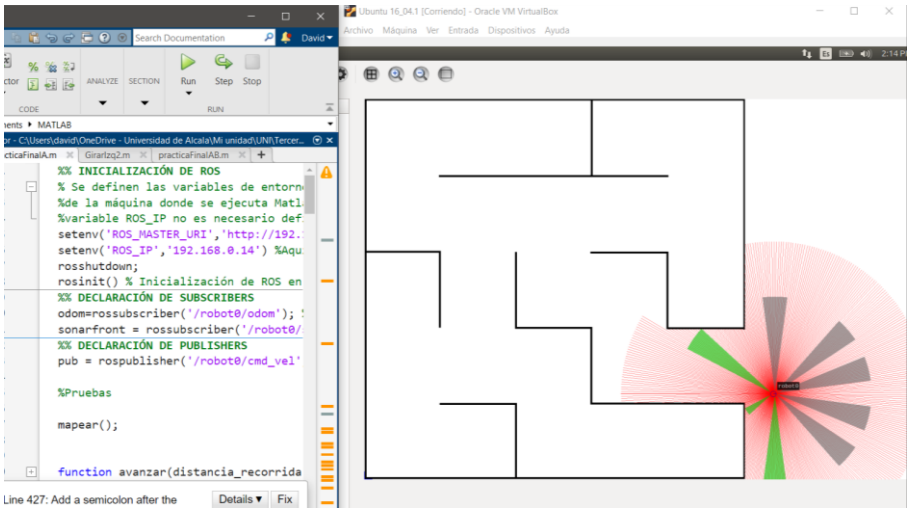
Aquí tiene tres celdas ya ocupadas la izquierda ocupada luego la siguiente prioridad es la derecha.

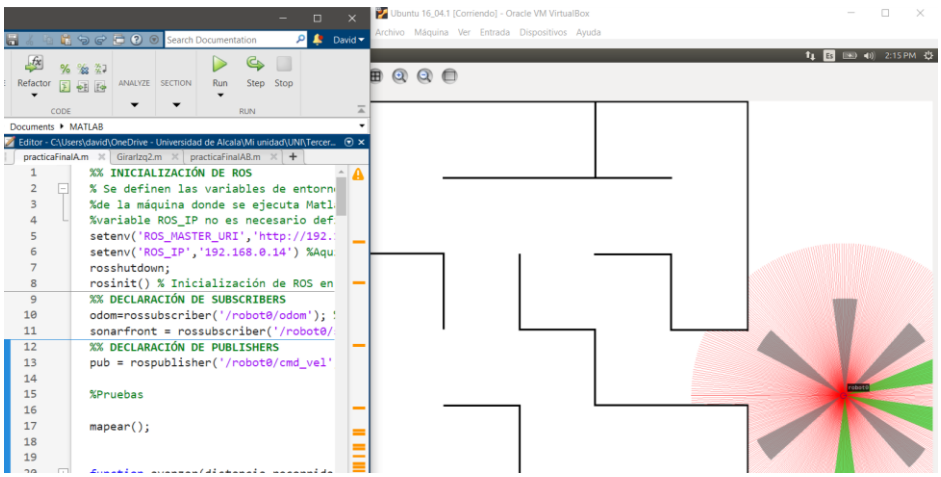

Search Documentation
David
Run
Step
Stop
RUN
practicaFinalAB.m
tion avanzar(distancia_recorrida)
tion paredes = detectarParedes()
tion girarder()
tion girarizq()
tion avanzarAtras()
girarder();
girarder();
avanzar(4.5);
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda


Search Documentation
David
Run
Step
Stop
RUN
practicaFinalAB.m
CIALIZACIÓN DE ROS
efinen las variables de entorn
máquina donde se ejecuta Matl
ble ROS_IP no es necesario def
('ROS_MASTER_URI','http://192.
('ROS_IP','192.168.0.14') %Aqu
tdown;
t() % Inicialización de ROS en
LARACIÓN DE SUBSCRIBERS
ssubscriber('/robot0/odom');
ront = rossubscriber('/robot0/
LARACIÓN DE PUBLISHERS
ospublisher('/robot0/cmd_vel'
as
();
on avanzar(distancia_recorrida
on paredes = detectarParedes()
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
width 30 m
height 20 m
olution 0.02 m/px


Search Documentation
David
Run
Step
Stop
RUN
practicaFinalAB.m
ICIALIZACIÓN DE ROS
definen las variables de entorn
a máquina donde se ejecuta Matl
able ROS_IP no es necesario def
v('ROS_MASTER_URI','http://192.
v('ROS_IP','192.168.0.14') %Aqu
utdown;
it() % Inicialización de ROS en
CLARACIÓN DE SUBSCRIBERS
rossubscriber('/robot0/odom');
front = rossubscriber('/robot0/
CLARACIÓN DE PUBLISHERS
rospublisher('/robot0/cmd_vel'
bas
r();
ion avanzar(distancia_recorrida
ion paredes = detectarParedes()
Ubuntu 16_04.1 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda







Ha recorrido 27 celdas, las celdas ya visitadas no cuentan como celdas y finaliza el programa

4. Conclusiones

El trabajo incluye varias funciones que trabajan en conjunto para permitir que un robot explore y mapee un laberinto utilizando el entorno de ROS y los datos de sensores. A continuación, se presenta una conclusión general que abarca todas las funciones:

El objetivo principal del trabajo es desarrollar un sistema robótico capaz de navegar y mapear un laberinto desconocido. Para lograr esto, se utilizan diferentes funciones que se encargan de tareas específicas.

La función girarizq() se encarga de girar al robot 90 grados hacia la izquierda. Esto es útil para cambiar de dirección y explorar nuevas áreas del laberinto.

La función girarder() realiza un giro de 90 grados hacia la derecha. Al igual que la función anterior, ayuda al robot a cambiar de dirección y explorar diferentes caminos.

La función avanzar() permite que el robot se desplace hacia adelante una distancia específica. Esto le permite explorar y cubrir distancias en el laberinto.

La función avanzarAtras() mueve al robot hacia atrás una distancia específica. Esta función puede ser útil cuando se necesita retroceder para explorar o cambiar de dirección en el laberinto.

Finalmente, la función mapear() combina todas estas funciones en un algoritmo de mapeo completo. Utilizando la información de los sensores y las decisiones de movimiento basadas en prioridades, el robot explora el laberinto y registra las paredes y casillas visitadas para construir un mapa del entorno.

En resumen, el trabajo aborda el desafío de la exploración y el mapeo de un laberinto mediante el uso de un sistema robótico controlado por ROS. Las funciones proporcionadas permiten que el robot se desplace, gire y registre información para construir gradualmente un mapa del laberinto. Este enfoque tiene aplicaciones prácticas en robótica, como la planificación de rutas y la localización en entornos desconocidos.

5. Bibliografía

Codigos de las anteriores practicas, funciones de las diapositivas proporcionadas por los profesores.