TRABAJO FINAL DE LA ASIGNATURA

GUIADO Y NAVEGACIÓN DE ROBOT LIMPIADOR EN APARTAMENTO

GUIADO Y NAVEGACIÓN DE ROBOTS

MÁSTER AUTOMÁTICA Y ROBÓTICA

Grupo 14

AUTORES

Javier Jiménez García

Jorge Gómez-Pamo

Tomas Sánchez Villaluenga

Índice

1. Introducción

En este proyecto se busca simular el comportamiento de un robot de limpieza del tipo Roomba, pero con un movimiento diferencial, por un espacio controlado. Dicho entorno es una simulación de un apartamento con 9 habitaciones decoradas con los muebles característicos de cada habitación.

La intención del proyecto es crear a partir del simulador Apolo un método de navegación de un robot limpiador que se desplace eficientemente por el entorno a partir de unas indicaciones globales dadas. Este robot conocerá previamente el mapa del apartamento, el nombre de cada habitación y la ubicación de cada una.

El proceso de limpieza comienza cuando el usuario por medio de la aplicación selecciona una habitación para limpiar. Entonces el robot se sitúa autónomamente en el punto de inicio de limpieza correspondiente a la habitación indicada, a través de sus métodos de localización y planificación.

2. Aplicación escogida

Como se ha indicado en la introducción, la aplicación escogida trata de la limpieza del suelo de un apartamento a través de un robot limpiador con movimiento diferencial.

El usuario con el uso de la aplicación elige una habitación para ser limpiada, el robot se acerca hasta ella evitando los obstáculos y se encarga de recorrer el área de la habitación haciendo un zig-zag. Una vez termina la tarea o el robot tiene poca batería se pone en camino de vuelta a la base de carga.

Robot Roomba de iRobot

2.1 La organización de las carpetas es la siguiente:

La carpeta 14_Calibración, el script para calibrar las matrices Q, P y R

La carpeta 14_Control, guarda las dos funciones de control, move_control, para cuando está haciendo la planificación y control2 cuando está haciendo el barrido.

La carpeta 14_FiltroKalman guarda la función que realiza el filtro de Kalman y devuelve la posición estimada

La carpeta 14_Imágenes guarda las imágenes de la aplicación

La carpeta 14_Incertidumbres guarda las funciones para obtener las incertidumbres de la odometría y la lectura de las balizas.

La carpeta 14_Mapas guarda los archivos xml con los mapas, MundoProyectoOBJETOS guarda el mapa con obstáculos y el mapa MundoProyectoSinObs guarda el mapa sin obstáculos.

La carpeta 14_Reactivo, guarda todas las funciones necesarias para el control reactivo durante la planificación y durante el barrido.

La carpeta 14_Voronoi guarda las funciones necesarias para obtener el grafo de desplazamiento por el mapa y la función A* para encontrar el camino por el grafo. Incluye un script titulado Voronoi.m, donde pueden comprobar la planificación en un bucle que va escogiendo puntos aleatorios dentro del mapa como inicio y fin.

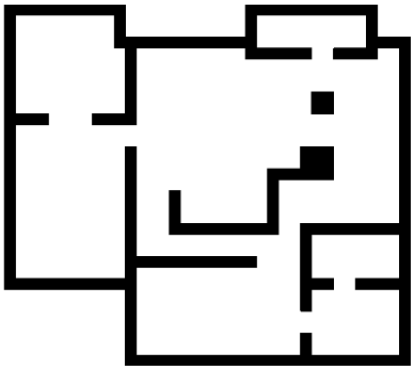
Finalmente está la aplicación Roomba_14, que al ejecutarla podrán hacer uso del robot y comandarle a qué habitación debe ir a limpiar. Y el video demostrador está en onedrive 14_Demostrador.mp4.

3. Mapa

En el caso del mapa se ha buscado simular lo que podría ser un apartamento con varias habitaciones, puertas y pasillos.

Para su creación se ha desarrollado un script de Matlab llamado "*mapaProyecto.mlx"* mediante el cual se pueden crear paredes y prismas de forma más intuitiva y automática, sin necesidad de tener que tocar los vértices de cada una de estas en el código *.xml* manualmente, lo que puede ser laborioso y dar lugar a muchos errores. Además, gracias a este script se ha podido escalar el mapa de la manera que se ajuste más a lo buscado.

Para la elaboración de la arquitectura se realizó primeramente un boceto que mostrase la forma que iba a tener el apartamento y sus dimensiones.



Croquis del mapa utilizado

Posteriormente se le aplicaros objetos dispuestos a lo largo del entorno, los cuales dan mayor complejidad a este.

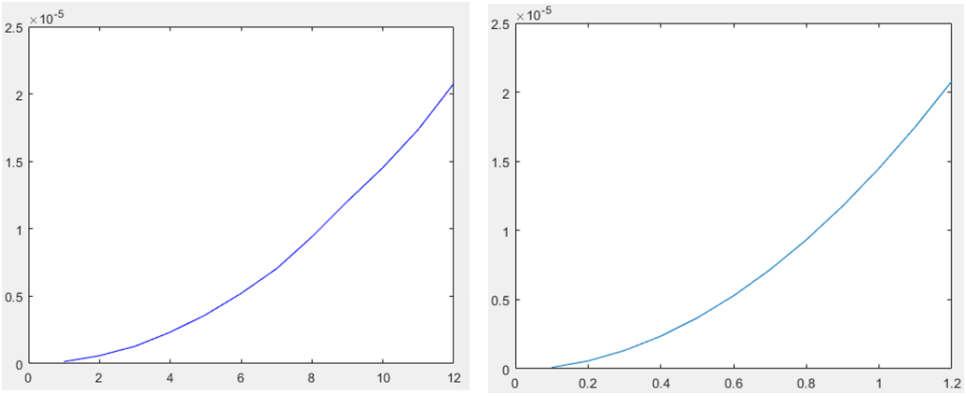
Mapa real utilizado

4. Sistema de locomoción

El robot se mueve con un sistema de locomoción diferencial, que para realizar un giro debe realizar un desplazamiento lineal.

4.1. Pruebas de calibración

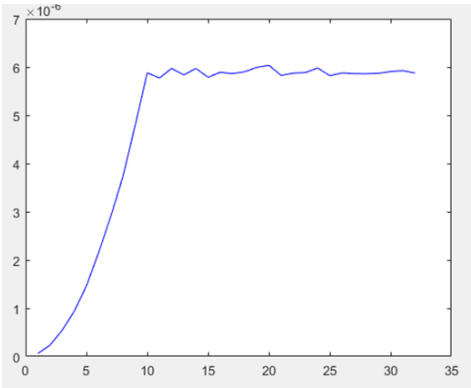
Se ha obtenido la incertidumbre de la lectura de la odometría. Esta lectura de la odometría está formada por la posición que se estima y la orientación que se estima. Para determinar la incertidumbre de posición lineal se ha situado el robot en una posición inicial y se han tomado diez mil muestras desplazando el robot con una velocidad y obteniendo la distancia entre la posición inicial y la posición que devuelve la odometría. Se ha realizado este experimento cambiando la velocidad y manteniendo el tiempo constante a 0.1, siendo este el valor que se esa en todo el proyecto. Se han hecho uso de velocidades desde 0.1 a 1.2 con incrementos de 0.1 y se ha obtenido la siguiente gráfica de incertidumbres habiendo tomado 10000 muestras de cada velocidad. A la izquierda está la gráfica de resultados y a la derecha la curva ajustada.

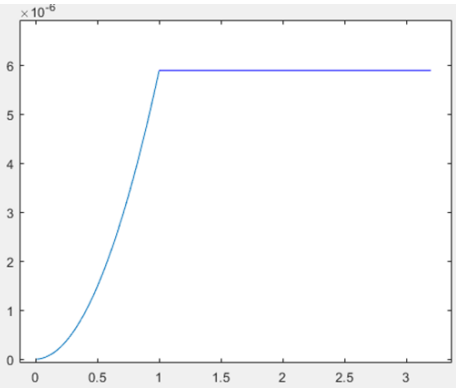


Como se puede observar la incertidumbre crece con la distancia recorrida y sigue una curva. Se ha ajustado una función de segundo grado a la gráfica, siendo esta la siguiente, De esta forma conociendo la velocidad lineal se puede obtener la incertidumbre asociada.

$$y = p1 * x^2 + p2 * x + p3, \quad \begin{matrix} p1 = 1.468 * 10^{-5} \\ p2 = -2.295 * 10^{-7} \\ p3 = 3.042 * 10^{-8} \end{matrix}$$

Para obtener la incertidumbre de posición angular se ha repetido el mismo experimento, pero esta vez la velocidad lineal se ha mantenido fija y se ha ido cambiando la velocidad angular desde 0.1 hasta 3.2 con incrementos de 0.1. Como se ve en la siguiente gráfica, la incertidumbre sigue una curva hasta llegar a una velocidad angular de 1 donde continua de forma constante. Se ha ajustado una curva a la primera porción de la gráfica y se ha obtenido la media de la segunda porción y se han obtenido los siguientes resultados, a la izquierda se ven los datos y a la derecha la curva ajustada.


×10^-5
7
6
5
4
3
2
1
0
0
5
10
15
20
25
30
35


×10^-5
6
5
4
3
2
1
0
0
0.5
1
1.5
2
2.5
3

5. Sensores escogidos

Se hace uso de dos sensores, el sensor de ultrasonidos y el sensor de telémetro en modo de detección de balizas reflectantes. El sensor telémetro detecta desde –135º a 135º tomando como 0º la dirección de avance frontal del robot. Se dispone de tres sensores de ultrasonidos que apuntan a 0º, -90º y 90º.

5.1. Pruebas de calibración

Para obtener la incertidumbre de la lectura de distancia y ángulo del sensor laser a las balizas se ha repetido el experimento, pero en vez de con una pared se ha hecho con una baliza. Se ha situado el robot a distintas distancias de la baliza manteniendo el ángulo y después se ha mantenido la distancia y se ha ido rotando el robot. Se realizaron diez mil lecturas en cada posición y se ha visto que la lectura de distancia del sensor laser no depende de la distancia y la lectura del ángulo no depende de la rotación del robot. Después se tomaron un millón de muestras desde una posición y se han obtenido las siguientes incertidumbres.

Incertidumbre de distancia es 1.9339e-4

Incertidumbre ángulo es 3.8637e-4

Para la incertidumbre de la lectura del sensor de ultrasonidos, el cual lee desde una distancia de cero metros hasta una distancia máxima de tres, se ha situado el robot delante de una pared, cambiando la distancia dentro del rango anterior y tomando diez mil muestras en cada distancia se ha obtenido una

6. Algoritmo de localización

Como algoritmo de localización se ha hecho uso del filtro de Kalman extendido. Como valores a medir para ajustar la predicción del filtro se toma el ángulo y la distancia a las balizas. Con estos valores de cada baliza se crea un vector con tamaño dinámico, este tamaño depende del número de balizas que se observa en cada momento.

La estimación de la z estimada del ángulo a partir de la posición del robot y el vector de la matriz H se obtienen con las siguientes formulas:

$$z = atan2\left(pos_{baliza_y} - pos_{odom_y}, pos_{baliza_x} - pos_{odom_x}\right) - pos_{odom_{theta}}$$

$$H = \frac{dz}{dx}$$

La obtención de la z estimada de las distancias con la posición del robot se hace con las siguientes formulas:

$$z = \sqrt{(\text{baliza}_x - odom_x)^2 + \left(baliza_y - odom_y\right)^2}$$

$$H = \frac{dz}{dx}$$

En cada movimiento se lee la odometría y se obtiene la nueva estimación

6.1. Experimentos realizados

Un aspecto muy importante a tener en cuenta en la localización es el análisis de la influencia de los principales parámetros del filtro de Kalman. Estos parámetros son P(0|0), Q(k) y R(k) y veremos cómo influye cambiar sus valores en los resultados obtenidos del filtro de Kalman.

En primer lugar, P(k) es la matriz de varianzas y covarianzas de la estimación del estado en el instante k y, por tanto, P(0|0) es la incertidumbre en el estado inicial de la estimación del estado. Este valor es clave en el filtro de Kalman ya que determina su posterior funcionamiento y puede conllevar un mal funcionamiento del filtro de Kalman por introducir un P(0|0) erróneo.

En segundo lugar, Q(k) es la matriz de varianzas y covarianzas de los sensores propioceptivos (en nuestro caso la odometría). Es imprescindible calibrar correctamente dicha matriz para un resultado óptimo en nuestro algoritmo de localización.

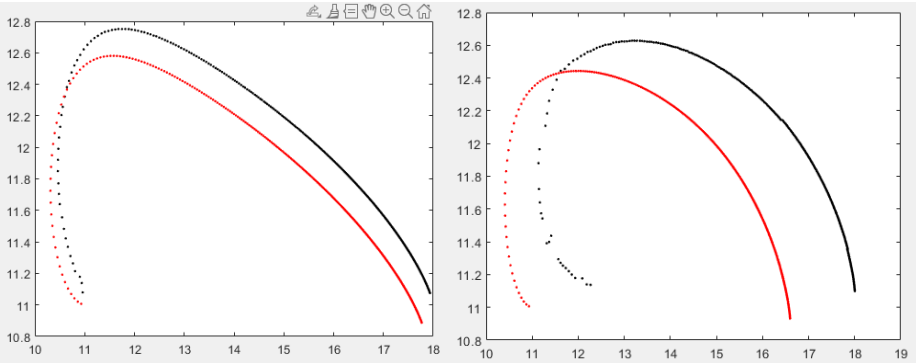
Por último, R(k) es la matriz de varianzas y covarianzas de los sensores exteroceptivos (sensores láser detectando las balizas) . Al igual que la matriz Q(k) es crítico calibrar de manera precisa sus valores.

Ahora analizaremos diferentes cambios en los parámetros expuestos anteriormente y valoraremos las consecuencias que suponen y su importancia.

La manera de cuantificar las diferentes pruebas será mediante el error cuadrático entre la posición real del robot medida mediante el comando apoloGetLocationMRobot y la posición estimada del filtro de Kalman en función de los sensores exteroceptivos (laser y balizas) y los sensores propioceptivos del robot (odometría).

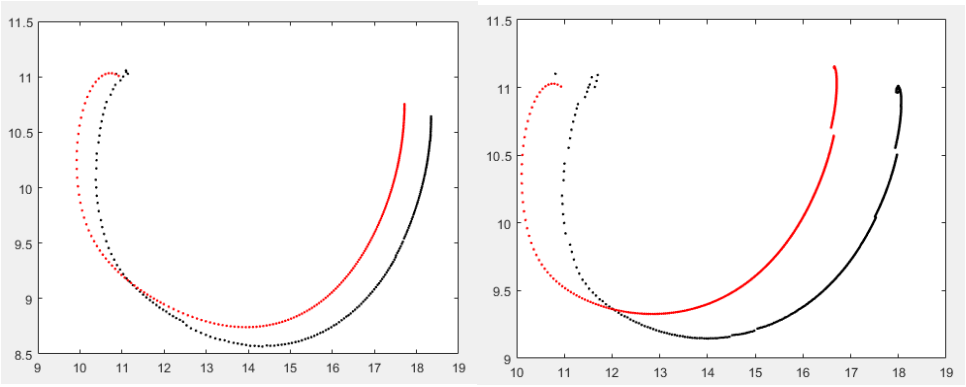
Para ello, variamos cada uno de los 3 parámetros de manera individual permaneciendo constante el valor de los otros dos durante las pruebas. Los resultados obtenidos difieren sustancialmente en función de la simulación alterando en el error cuadrático, pero se tomará una media de las medidas de error cuadrático obtenidas.

En el caso de P(0|0), con unos valores ajustados de Q(k) y R(k), no se aprecia una gran diferencia en el error cuadrático alterando los valores entre 1, 0.1, 0.01, 0.001 y 0.0001. Esto puede ser debido a que al estar bien ajustadas las matrices de los sensores, la estimación en el estado inicial no es crítica. A pesar de ello, sabemos que en un principio P(0|0) no debe ser muy elevado y que es preferible tomar un valor más alto del óptimo que uno inferior.



Resultados obtenidos de la trayectoria para P(0|0) ajustado (izda) y para uno no ajustado (dcha)

Para la matriz Q(k), una reducción de sus valores implicaría que las medidas que el robot obtiene de la odometría tendrían un error más cercano a cero. Analizando el error cuadrático, reducir significativamente los valores de Q hace que el error cuadrático aumente debido a que las lecturas de odometría no tienen una precisión tan alta como los valores de varianzas que hemos introducido. Por el contrario, si subimos enormemente sus valores, el robot piensa que las medidas de odometría no son buenas y les dará más peso a las medidas de los sensores exteroceptivos. Por lo tanto, se trata de encontrar un equilibrio en función de la bondad de las medidas que obtiene el robot de su odometría.



Resultados obtenidos de la trayectoria para valores de Q(k) ajustado (izda) y para uno no ajustado (dcha)

Por último, la variación de la matriz R(k) tiene unas consecuencias equivalentes a las de la matriz Q(k) pero dando mayor o menor importancia a las medidas exteroceptivas en este caso. Para esta matriz, los resultados han indicado que los valores de R(k) han de ser inferiores a los valores de Q(k) debido a que las medidas obtenidas de los sensores láser y de las balizas tienen una mayor precisión (menor incertidumbre) que las medidas de la odometría.

Una vez analizados los efectos de manera individual, realizaremos diferentes pruebas alterando los valores de P(0|0), Q(k) y R(k) hasta encontrar los valores que mejores resultados obtengan en términos de error cuadrático.

Los resultados obtenidos en las pruebas son los valores finales que se han ido mostrando a lo largo de la memoria un poco alterados.

P(0|0) = 0.0001

R(1,1) = 1.8578e-04
R(2,2) = 3.7847e-04

Parámetros p1, p2 y p3 para el ángulo (Q(k)):

p1a = 6.021e-06

p2a = -9.999e-08

p3a = 5.9123e-06

Parámetros p1, p2 y p3 para la distancia (Q(k)):

p1d = 1.4891e-04

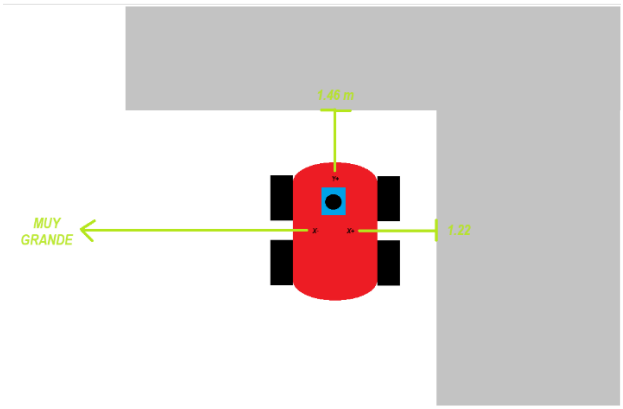
p2d = -2.367e-07

p3d = 3.212e-08

7. Algoritmo de control reactivo

Para la creación del algoritmo de control reactivo del robot se ha tenido en cuenta la regularidad de las paredes y obstáculos del mapa, las dimensiones y el movimeinto diferencial del robot, la colocación de los sensores de ultrasonido y la disposición de los obstáculos en el entorno.

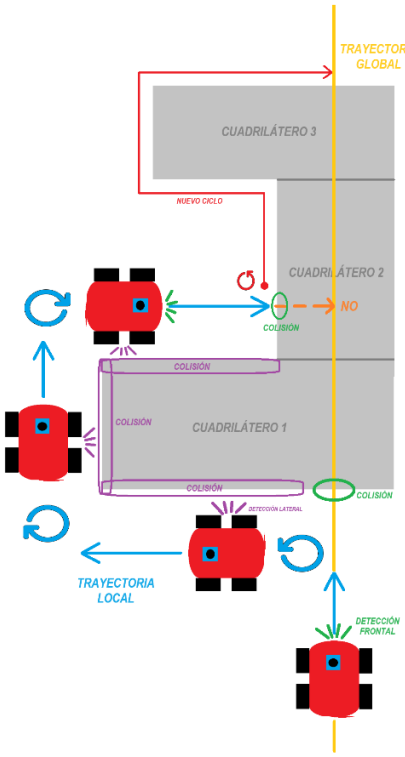
Debido a que el barrido del robot sigue trayectorias verticales en valores constantes del eje X, el algoritmo implementado lo que hace es bordear el obstáculo por su lado izquierdo hasta llegar de nuevo a la vertical de su trayectoria global. Esta trayectoria local de bordeado se realiza con ayuda de los sensores ultrasonidos dispuestos en el robot, los cuales evalúan la distancia existente a obstáculos en los ejes Y positivo, X positivo y X negativo del robot, siendo el eje Y positivo la orientación frontal de este.

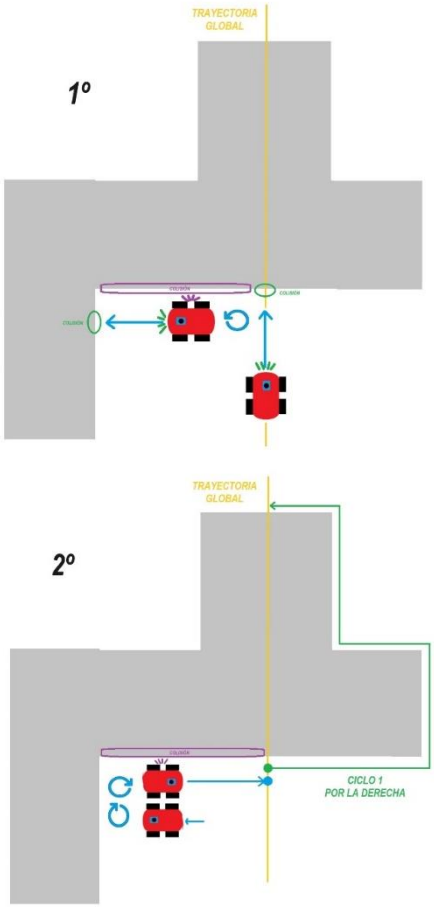


Evaluación de distancias por ultrasonidos

Se han creado dos tipos de algoritmos reactivos, los cuales pueden implementarse individualmente o pueden también complementarse. El primer algoritmo se compone de 2 únicos ciclos que se realizan tras la evaluación de existencia de un obstáculo en la trayectoria. El primer ciclo es el descrito anteriormente. Este ciclo busca rodear el obstáculo por el lado izquierdo del robot y evoluciona cuadrilátero a cuadrilátero. Esto quiere decir que bordea el primer cuadrilátero buscando la vertical de su trayectoria global y, en el caso de que tras el bordeado no le permita retomar dicha trayectoria, vuelve a realizar este mismo ciclo bordeando de nuevo el obstáculo, ya que esto quiere decir que la base de dicho obstáculo se compone de varios cuadriláteros. El avance se ve guiado a partir de los valores de los sensores laterales del robot, a partir de los cuales sabe en todo momento de la existencia de obstáculo. Una vez el sensor detecta que ha evadido el obstáculo lateral, este avanza y gira hacia la orientación en la que se encuentra el obstáculo para seguir bordeándolo satisfactoriamente. Este ciclo se repetirá hasta que el robot encuentre la vertical que dicta su trayectoria global o evalúe que ha llegado al límite superior o inferior de la habitación donde se encuentra.

El segundo ciclo se basa en el anterior, pero este se ha implementado para solucionar el caso en el que el rodeo del obstáculo por la izquierda del robot está impedido, ya sea porque existe otro obstáculo o una pared que lo inhabilite. En este caso, lo que hace el robot es, tras identificar el impedimento, realizar un giro de 180º hasta volver a la vertical de donde proviene y realizar el bordeado del obstáculo por la derecha de la misma manera que el ciclo anterior, pero cambiando el sentido de sus giros.





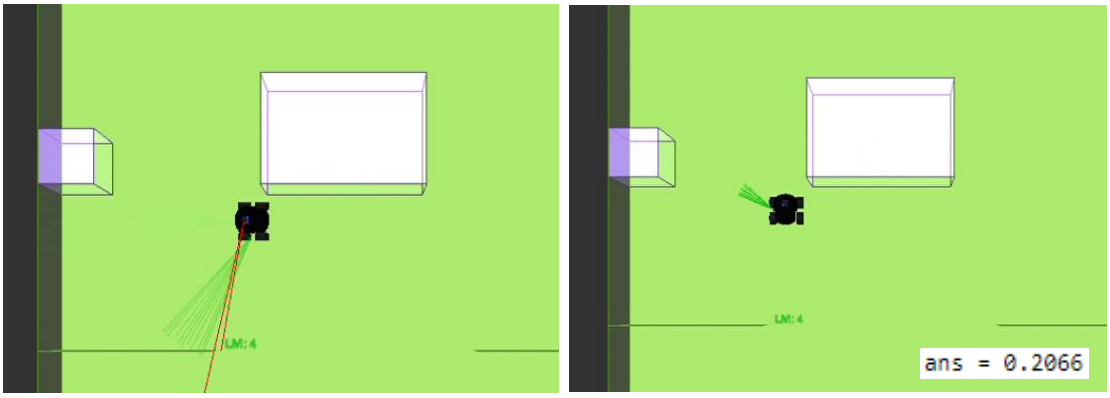
Proceso realizado por el ciclo 1 *Proceso realizado por el ciclo 2*

Por otro lado, el segundo algoritmo de control reactivo busca esquivar obstáculos en vez de rodearlos. Está pensado principalmente para aplicarse junto al avance del robot cuando este sigue las trayectorias generadas a través de Voronoi. Este se basa en la comprobación de la geometría que rodea al robot. A través de los valores aportados por los sensores de ultrasonidos, el robot evaluará que hacer en los siguientes casos de esta manera:

- *Obstáculo izquierdo* → alejamiento lento hacia la derecha.
- *Obstáculo derecho* → alejamiento lento hacia la izquierda.
- *Obstáculo frontal* → gira hacia la dirección que no haya más obstáculos (izqda. por defecto).
- *Obstáculo izquierdo y frontal* →gira hacia la derecha tras realizar una marcha atrás leve.
- *Obstáculo izquierdo y derecho* → avanza recto con una velocidad angular pequeña en función de qué obstáculo se encuentra más cercano al robot.
- *Obstáculo derecha y frontal* →gira hacia la izquierda tras realizar una marcha atrás leve.
- *Obstáculo derecha, izquierda y frontal* → marcha atrás brusca seguido de un giro.

7.1. Experimentos realizados

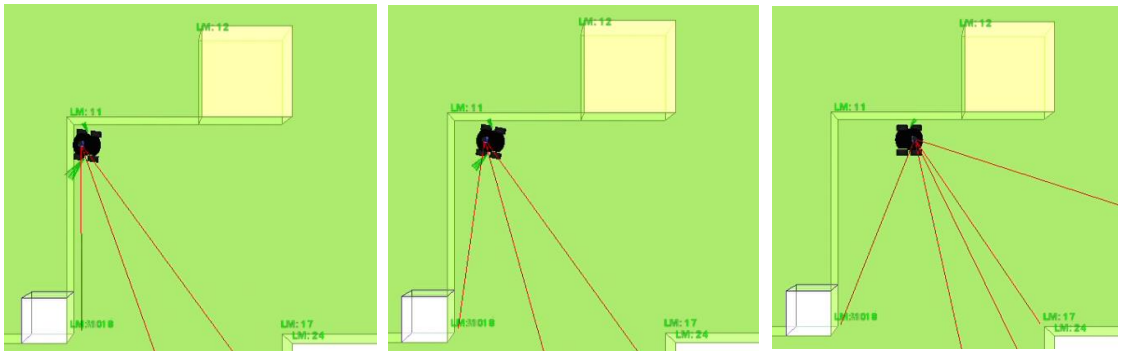
Para la comprobación del correcto funcionamiento del algoritmo se ha situado el robot en diferentes sitios del mapa y se ha simulado su comportamiento con los distintos tipos de obstáculos existentes en el entorno. Tras varias pruebas de los distintos ciclos se tuvo que modificar el algoritmo, ya que se vio que existía la necesidad de que el robot avanzase una pequeña distancia frontalmente antes de girar a la que se ha llamado *"distancia_Offset"*. Esto es debido a que, tras el giro, el robot puede seguir sin evaluar que existe un obstáculo, por lo que se dejaría de funcionar el control reactivo debido a una mala identificación del obstáculo.



Error en el giro al rodear el obstáculo

Como se puede ver, el valor de distancia detectado por el sensor frontal (distancia = 0.2066 m) es menor al límite preestablecido (límite = 0.3 m), por lo que el robot no avanza y se queda situado en esa misma posición.

Por otro lado, en el caso del ciclo 2, se tuvo que implementar un desplazamiento de retroceso con respecto a la frontal, ya que el segundo giro que evalúa la colisión con la segunda pared provocaba que el robot se acercase demasiado a la pared y no se le permitiese rotar sobre su eje.

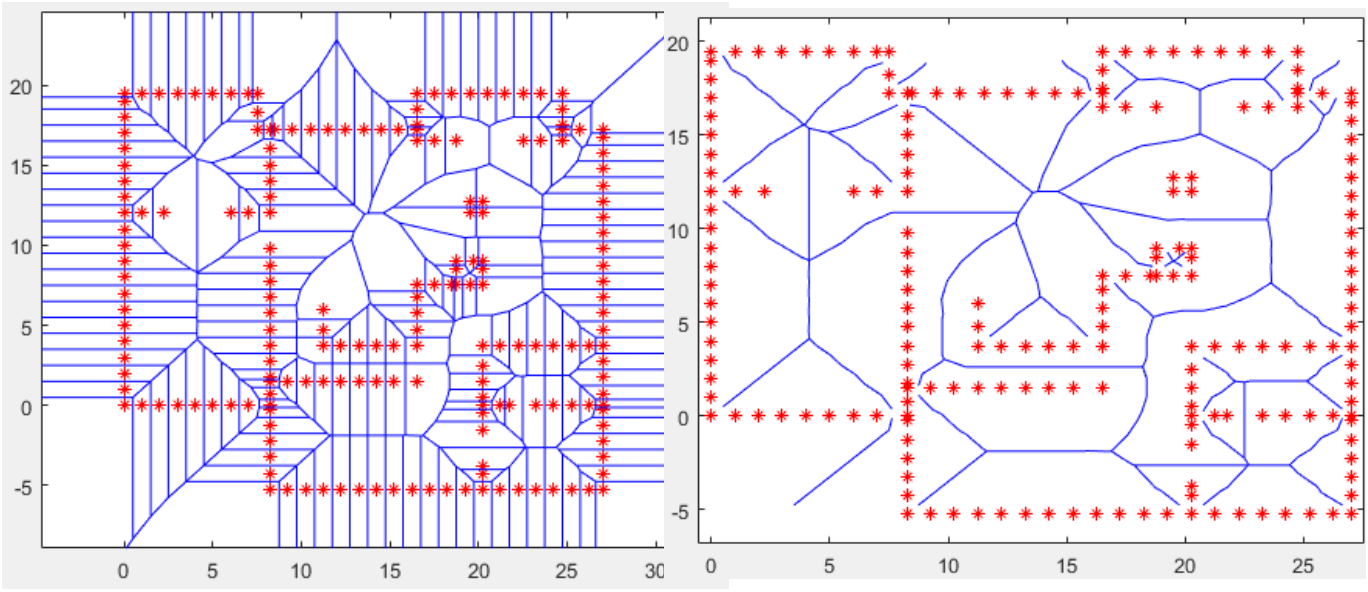


(a) Robot bloqueado ; (b) Marcha atrás ; (c) Retoma la trayectoria

Todos los movimientos del control reactivo realizan una evaluación de choques, mediante la cual, gracias a la función *apoloMoveMRobot()*, si existe el caso de que el robot sufre un choque, se le aplica un retroceso para que pueda movilizarse de nuevo.

8. Algoritmo de planificación

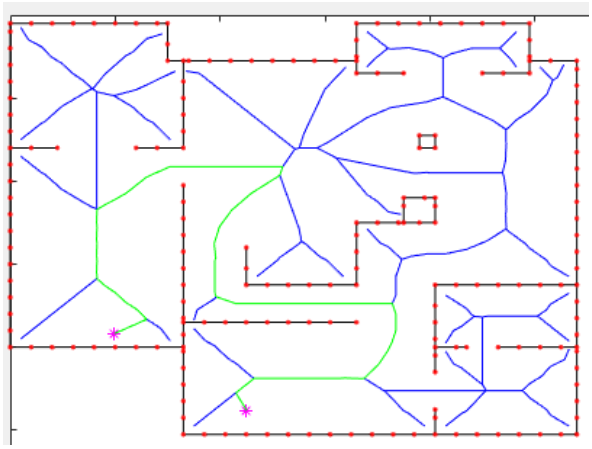
Para la planificación se ha hecho uso de voronoi para obtener los caminos de mayor distancia con las paredes. Todas las funciones relacionadas con la planificación, están en la carpeta Voronoi. Para ellos se ha guardado en un archivo Excel titulado VerticesVoronoi() las coordenadas de los finales de las rectas que componen el mapa. Estas rectas han sido cortadas en trozos y con el uso de la función voronoi() se han obtenido las líneas que muestran los espacios geométricos de todos estos puntos. A continuación, se han filtrado todas las rectas que intersecan las paredes y se ha obtenido el siguiente resultado. A la izquierda se muestra el resultado sin filtrar y a la izquierda filtrado

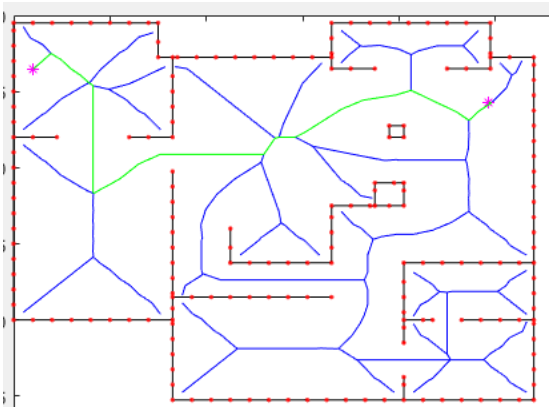


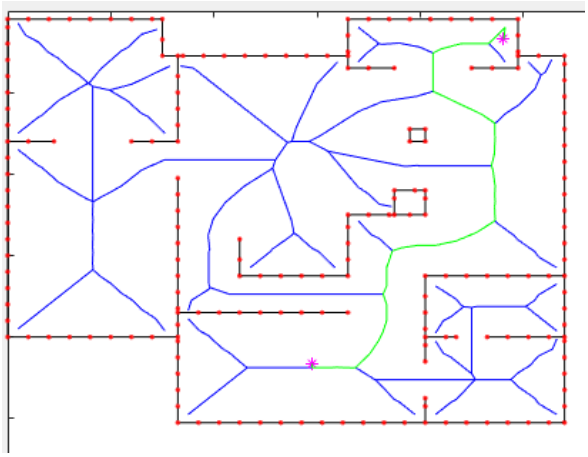
Para hacer uso de la información obtenida, se ha convertido en un grafo donde cada nodo tiene como información, su posición en el mapa y los nodos a los que está unido. Para obtener la ruta de un punto inicial a uno final, primero se busca el nodo más cercano al punto inicial asegurando que si el robot va en línea recta hasta el punto no se chocaría con ninguna pared, de eso se encarga la función

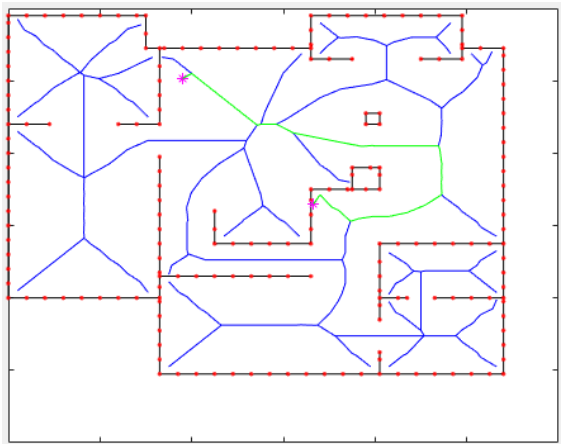
8.1. Experimentos realizados

Se ha creado una función que crea el grafo como se ha explicado previamente y después se escogen dos puntos aleatorios asegurándose que están dentro del mapa y se realiza el A* para llegar de un punto a otro, esto se ha repetido en bucle durante horas y se ha observado que siempre se ha conseguido correctamente unir el punto inicial y el final. A continuación, se muestran varios ejemplos. Se observa como el camino resultante es el camino que más se aleja de todas las paredes y el más optimo posible con esta condición.









9. Controlador

Como controlador se hace uso de un controlador proporcional tanto para la velocidad lineal como para la velocidad angular. Para la velocidad lineal se multiplica la diferencia entre la posición estimada y la posición a la que se quiere llegar por una contante. Si el resultado tiene un valor superior de 1.2 se reduce a 0.8, para así evitar velocidades muy grandes. Para la velocidad angular se obtiene el ángulo que necesita tener el robot para mirar a la posición, esto se hace obteniendo el ángulo que hay entre la posición estimada y la posición deseada, en la siguiente formula se muestra:

$$angulo = atan2\left(pos_{deseada_y} - pos_{estimada_y}, pos_{deseada_x} - pos_{estimada_x}\right)$$

Se determina cual es el sentido que requiere el menor giro para llegar a esta posición y se multiplica el giro menor por una contante. Después se le suma la velocidad lineal que se obtuvo por una constante, para así, si se desplaza mucho también aumente en la misma proporción el ángulo que gira.

De esta forma el robot gira a la vez que se desplaza, ya que, al ser un robot diferencial, no puede girar si no se desplaza a la vez.

Además, se ha implementado un segundo controlador para el barrido de limpieza con la misma estructura que éste, pero con unos valores diferentes de los parámetros para poder realizar unos movimientos más adecuados a la aplicación de limpieza en habitaciones. De esta forma, la limpieza es uniforme y a una velocidad adecuada a una Roomba del mercado actual.

10. Barrido de limpieza

La funcionalidad de nuestro robot es ofrecer un servicio de limpieza en un entorno controlado de un apartamento con 9 diferentes habitaciones con sus diferentes muebles (para el robot todos esos muebles son obstáculos).

Para ello se ha implementado un algoritmo de limpieza que permite realizar el barrido en las nueve diferentes habitaciones del entorno. En la interfaz de la aplicación diseñada el usuario puede seleccionar libremente cualquiera de las 9 habitaciones para que realice la limpieza de la misma. De esta forma y, ayudado del planificador, el robot se desplazará al punto de inicio de limpieza de la habitación escogida. Una vez el robot se encuentra en el punto de inicio comienza el algoritmo de barrido.

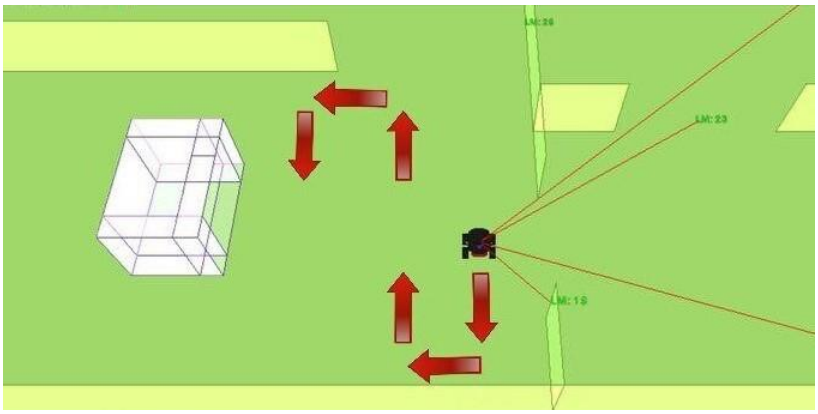
Mapa del apartamento con los puntos iniciales de barrido de color azul, el punto de carga del robot en rojo y el nombre de las diferentes habitaciones

En la figura superior se pueden observar las diferentes habitaciones junto a sus nombres. En el desarrollo de la aplicación se han definido las habitaciones numeradas del 1 al 9. Para el usuario que realice la limpieza de este apartamento se define la correspondencia entre los números de las habitaciones y sus nombres.

Habitación 1 – Entrada	Habitación 4 - Pasillo	Habitación 7 -Dormitorio 1
Habitación 2 – Cocina	Habitación 5 – Despensa	Habitación 8 – Dormitorio2
Habitación 3 – Salón 1	Habitación 6 – Comedor	Habitación 9 – Salón 2

El algoritmo de barrido ha sido diseñado para recorrer las habitaciones verticalmente desplazándonos en el eje ‘y’ de nuestro entorno. Inicialmente, el robot se desplazará verticalmente hasta el límite inferior o superior de la habitación dependiendo de si comienza en valores de ‘y’ mayores o menores, respectivamente. Una vez realizado el barrido vertical, el robot se desplazará horizontalmente unas 0.5 unidades de medida(¿) y continuará con el barrido en el sentido vertical contrario a su anterior movimiento. El

ancho de desplazamiento se ha considerado adecuado debido a las dimensiones del robot y a la posibilidad de poder realizar la limpieza correctamente con dicho valor. De esta forma, el robot es capaz de realizar el barrido por toda la habitación recorriéndola verticalmente y de manera sistemática.



Es importante recalcar, que durante el barrido por las diferentes habitaciones se hace uso en paralelo del control reactivo para poder evitar diferentes obstáculos y continuar con la limpieza de la habitación de la manera descrita previamente.

El barrido de limpieza lleva incorporado un segundo controlador adicional al controlador para la planificación de tal forma que los valores de las velocidades lineales y angulares se ajustan más fielmente a una limpieza de una Roomba del mercado. Este controlador hace que se reduzcan las velocidades cuando se va a realizar el barrido horizontal y en el barrido vertical la velocidad sea lo suficientemente reducida para poder limpiar correctamente la habitación.

Una vez completada la limpieza de la habitación escogida por el usuario, el robot de manera automática va a desplazarse al punto de carga para poder aumentar el valor de su batería. Esta acción puede ser interrumpida en cualquier momento introduciendo una nueva habitación para limpiar. A medida que se va realizando la limpieza de una habitación la batería va reduciéndose progresivamente hasta cuando alcanza un valor del 10% que el robot interrumpe la limpieza y se dirige de manera inmediata al punto de carga.

11. Descripción del demostrador final

En el demostrador se ve como el robot empieza en la estación de carga y el usuario le da la orden de limpiar la habitación dos, el robot se desplaza hasta la habitación dos, con el uso de la planificación y evitando los obstáculos. Cuando llega a la habitación empieza a recorrer toda la habitación para limpiarla haciendo un movimiento de zig-zag. Una vez que ha terminado vuelve a hacer uso de la planificación y localización para volver a la estación de carga. Es importante ver que en todo momento el usuario puede ver la velocidad del robot y se observa como la batería va reduciéndose en función del tiempo y la velocidad del robot. Por último, se ve que cuando vuelve a la base se empieza a cargar y la batería sube de porcentaje y se enciende la bombilla.

12. Reparto de roles

Tareas	Encargados
Creación del mapa	Tomás
Moldeado de incertidumbres	Jorge, Javier y Tomás
Algoritmo de localización	Jorge y Javier
Algoritmo de control reactivo	Tomás
Algoritmo de barrido	Javier
Algoritmo de planificación	Jorge
Controlador	Javier
Demostrador final	Tomás y Jorge

13. Conclusiones

El trabajo ha requerido la utilización de un robot diferencial que ofrece ciertas ventajas para tareas de navegación libre, pero tiene algunos inconvenientes para tareas como la empleada en el trabajo. La mayoría de las Roomba del mercado son robots omnidireccionales que pueden rotar sobre sí mismo y facilitan enormemente la tarea de limpieza de las habitaciones y la evasión de posibles obstáculos en la limpieza.

Al utilizar un robot diferencial nos hemos encontrado dificultades añadidas referidas al giro del robot en los barridos de limpieza y al esquivar obstáculos que se iba encontrando en el camino. El haber podido utilizar el robot Marvin como robot omnidireccional (en Apolo es capaz de girar sobre sí mismo sin desplazarse) habría facilitado la realización de la limpieza y el control reactivo simultáneo.

El control reactivo también ha sido un desafío enorme debido a la dificultad de evitar objetos con el sistema de locomoción diferencial debido a la infinidad de formas que puede tener un obstáculo.

A pesar de ello, el Robot puede realizar sus barridos de limpieza correctamente y esquivar los objetos añadidos en el apartamento mediante el control reactivo. Además, cuenta con una interfaz para el usuario y un nivel de batería y un puesto de carga que ofrecen un servicio cercano a la realidad e ideal para cualquier cliente.