UNIVERSIDAD
POLITÉCNICA
DE MADRID

INDUSTRIALES
ETSII | UPM

GUIADO Y NAVEGACIÓN DE ROBOTS

TRABAJO DE LA ASIGNATURA

Hugo Rajado Sánchez M19205

Adrián Caro Zapata M19032

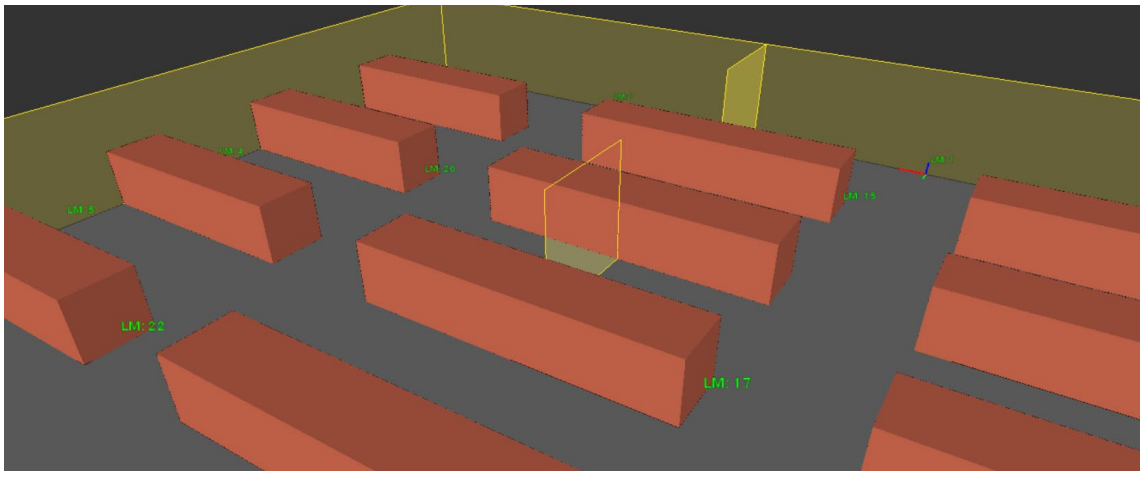
Pablo Dopazo Cordones M19066

ENERO 2020

Master en Robótica y Automática | Universidad Politécnica de Madrid | ETSII

Trabajo realizado por Hugo Rajado Sánchez, Adrián Caro Zapata y Pablo Dopazo Cordones para la asignatura de Guiado y Navegación de Robots, impartida por los profesores Fernando Matía, Miguel Hernando y Paloma de la Puente.

Implementado usando MATLAB 2019a y Apolo Simulator.



ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. Planteamiento del ejercicio

Para este trabajo, se plantea el diseño de un robot, capaz de transportar un objeto desde un punto origen en la oficina hacia un punto destino en el almacén. Para esto deberá recorrer todas las instalaciones, pasando a través de varias habitaciones y pasillos. Una vez en el almacén, el robot deberá navegar entre el entramado de estanterías hasta el punto de destino, donde supuestamente depositará la carga.

Para ello, contará con ayuda de un planificador RRT-Connect, que generará una ruta de navegación. Con un algoritmo de localización configurado tras una calibración de sensores, el robot irá ajustando su localización mediante la odometría y balizas detectadas por láser para seguir la ruta, a la vez que avanza gracias a un controlador de los motores. Este controlador puede ver interrumpida por la función reactiva, que evitará que colisione con objetos gracias a tres sensores ultrasónicos situados en su parte frontal.

1.2. Reparto de roles

Gracias a la disponibilidad y capacidad de todos los componentes del grupo de reunirnos para desarrollar el trabajo, decidimos trabajar todos los aspectos de este en conjunto. Esta decisión quizás alargó ligeramente el desarrollo del proyecto, pero nos permitió una mejor capacidad de aportar ideas y de apoyarnos mutuamente ante determinados retos. Además, obtuvimos una mejor comprensión del trabajo en su totalidad, que creemos que es el objetivo final de la asignatura. En otras palabras, todo el trabajo lo hicimos de forma conjunta por lo que la aportación a este ha sido equitativa.

2. FUNCIONES Y SCRIPTS DE MATLAB

2.1. Función configuración.m

Carga todos los parámetros necesarios para poder ejecutar todas las otras funciones de MATLAB y simulaciones de Apolo en el 'workspace' de MATLAB.

2.2. Función choque.m

Esta función recibe un punto (x, y) y devuelve un valor booleano: 1 si no choca, 0 si choca.

Para comprobar si choca, primeramente, se realiza un posicionamiento del robot con la función 'apoloPlaceMRobot' para comprobar la colisión en ese punto.

Para detectar si choca con una pared en una región próxima de movimiento, se ha utilizado la función 'apoloMoveMRobot' ya que devuelve un 0 si choca. Para un mejor resultado, se ha elegido que solo se mueva de forma lineal, y que el robot se oriente cada $\pi/3$ radianes en un rango de $[-\pi, \pi]$, comprobando si choca en todas esas direcciones. Esta última función nos permite crear un margen de seguridad respecto a las paredes ya que se puede configurar la distancia que se mueve el robot comprobando si choca mediante el ajuste de su velocidad y tiempo de ejecución del movimiento, descartando puntos demasiado próximos a las paredes que puedan causar colisión dada las dimensiones del robot.

2.3. Función nearest.m

Recibe como parámetros un punto (x, y) y una estructura árbol. Va uno a uno recorriendo los puntos del árbol y calculando su distancia respecto al punto proporcionado. Devuelve el punto del árbol más cercano a dicho punto y su índice (posición) dentro de la estructura árbol.

2.4. Función RRT.m

Genera puntos aleatorios e intenta conectar con el punto más cercano de un árbol que tiene como inicio el punto origen. Discretiza la recta de unión añadiendo los puntos que no chocan. En el caso de que algún punto choque, añade el segmento que lo precede al árbol y vuelve a crear otro punto aleatorio. El algoritmo se repite un número configurado de veces.

Se establece que cada cierto número de iteraciones, en vez de con un punto aleatorio, el árbol se intente conectar con el punto de destino. En el caso de conseguirlo, realiza el 'backtracking' (función 'backtracking.m') y devuelve como parámetro una vector de puntos solución desde el origen al punto final.

2.5. Función RRT_CONNECT.m

Se basa en la misma lógica que el algoritmo RRT, pero utilizando dos árboles, uno nace del nodo origen y otro nace del nodo destino. Intentan encontrarse según la siguiente secuencia, que se repite un numero configurado de veces hasta que los árboles consigan unirse al mismo punto.

- Árbol 1(origen) intenta unirse a un punto aleatorio.
- Árbol 2 (destino) intenta unirse al último punto añadido al árbol 1.
- Árbol 2 intenta unirse a un punto aleatorio.
- Árbol 1 intenta unirse al último punto añadido al árbol 2.

Una vez ambos arboles se encuentran, mediante la función backtracking.m se encuentran su vector solución, y estos se concatenan de forma ordenada, devolviendo una única matriz de puntos solución.

2.6. Función backtracking (backtracking_RRT.m, backtracking_RRTCONNECT.m)

Recibe una estructura árbol y su punto de origen. Devuelve el vector de puntos solución, ordenados desde el origen al nodo final.

Empieza en el punto final del árbol, guardando el punto en un vector auxiliar, y dirigiéndose a su punto padre (posición referenciada en su componente "padre" de la estructura). Lo repite sucesivamente hasta llegar a un punto cuyo padre sea el nodo inicial. Finalmente se añade el nodo inicial al vector auxiliar.

Dado que el vector auxiliar va de final a inicio, se procede a darle la vuelta para obtener un hilo desde el punto de origen hasta el punto final.

En el caso del 'backtracking' para el RRT-Connect, no se realiza el paso de darle la vuelta al vector, ya que esto va a depender de si se trata del árbol origen o el árbol destino. La inversión de este hilo se realiza de forma externa en la propia función del RRT-Connect.

2.7. Función smoothing.m

Recibiendo un vector de puntos (x, y) y un valor de incremento, realiza 500 iteraciones eligiendo dos puntos aleatorios del vector de puntos de entrada e intenta unirlos mediante una recta. Esta recta se discretiza utilizando según el incremento dado. Si en ese intento de unión, algún punto creado choca (según la función anterior), se aborta el intento de unión y se pasa a la siguiente iteración, probando con otros dos puntos aleatorios del vector. En el caso de que no choque ningún punto en el intento de unión, se reemplazarán los puntos intermedios del vector por los nuevos de la recta creada.

2.8. Función controlador.m

Envía los comandos de velocidad y giro al simulador de Apolo en función de la posición y orientación actual del robot (información proporcionada por la función de localización) y las distintas posiciones que debe ir alcanzando, dada la ruta establecida por el planificador.

La metodología se basa en los siguientes pasos:

1. Gira el robot hasta alinear su eje de movimiento con el punto de destino siguiente, con una cierta tolerancia.
2. Calcula la distancia hasta ese punto y ejecuta los comandos de movimiento lineal necesarios mediante la función 'apoloMoveMRobot'.
3. Repite el bucle para el siguiente punto de la ruta hasta que alcanza la posición de destino.

En el caso de que se cumplan ciertas condiciones de proximidad en función de las medidas de los sensores de ultrasonidos, llama a la función encargada del control reactivo, interrumpiendo el bucle anterior. Estas condiciones son las siguientes:

- Que la medida del sensor ultrasónico central sea inferior a 0.2
- Que la medida del sensor ultrasónico izquierdo sea inferior a 0.2
- Que la medida del sensor ultrasónico derecha sea inferior a 0.2

2.9. Función reactivo.m

Esta función la llama el controlador principal cuando los sensores de ultrasonido detectan que el robot está cerca de colisionar. Entra en un bucle que bloquea el avance lineal y procede a rotar en el sentido correspondiente hasta que los tres sensores dejen de detectar un obstáculo próximo. Al ocurrir esto, vuelve al cuerpo principal de la función controlador y salta a la iteración del bucle que corresponda con el punto de la ruta más cercano a la localización estimada actual. Es decir, su nuevo punto objetivo será el más cercano de la ruta a la localización del robot.

2.10. Función localizaciónDistancia.m

Esta es una implementación iterativa del Filtro Extendido de Kalman, es decir, un estimador dinámico que en este caso utiliza las medidas de la odometría del robot y la distancia a las balizas que le sean visibles. La función recoge la matriz de covarianzas del estado P_k y el estado X_k del momento anterior y devuelve sus valores para el momento actual.

2.11. Script calibracion.m

El script de calibración se encarga de realizar una serie de pruebas para comprobar la fiabilidad de las medidas hechas por los sensores del robot. Dicho script está compuesto por cuatro módulos, que el usuario puede ejecutar por separado, para llevar a cabo pruebas correspondientes a la calibración de la odometría lineal y angular, a la calibración de los sensores láser y a la calibración de los sensores de ultrasonidos.

2.12. Script main.m

Este es el código central de la implementación. Desde aquí se interacciona con el resto de las funciones, se hacen las comprobaciones necesarias, se lleva el control de flujo del programa y se declaran las pausas necesarias para garantizar la correcta sincronización con los procesos de la simulación en Apolo. Para lanzar los algoritmos y seguidamente la simulación de Apolo se deben configurar los parámetros en el archivo 'configuración.m' y ejecutar esta script.

3. ENTORNO

El mapa (mapa.xml) consta de dos sectores claramente diferenciables, una parte que simula una zona de oficinas, con paredes finas y con salas, y otra en configuración de almacén, cuyas estanterías forman un entramado.

Conociendo la metodología de actuación de nuestro planificador RRT-Connect, se decidió añadir una parte que pusiese en compromiso su efectividad. En la parte izquierda, en el sector de oficinas, se diseñó un pasillo en forma de 'U' invertida, de forma que el planificador afrontase su intento de conectarse con el nodo más cercano del otro árbol, viéndose frustrados sus intentos de conexión por una delgada pared y teniendo que esperar al 'azar' de la generación de un punto aleatorio en la parte superior para poder salir de esa dificultad.

El entramado de la zona del almacén crea numerosos giros y da dinamismo al planificador, ya que puede alcanzarse un punto por distintas rutas. A menudo el equipo se ha encontrado como la función de 'smoothing.m' ha conseguido acortar la ruta en el sector del almacén llevando la ruta solución por otro camino más corto y directo.

Las balizas se han situado intentando que, al menos, el sensor del robot siempre visualizase dos balizas, aunque el óptimo sería la visualización de tres balizas. Adicionalmente, se añadieron balizas al final de cada pasillo sin salida, ya que en este el robot quedaría ciego si entrase en él. La posición de balizas cercanas ha tratado de ofrecer la mayor diferencia de ángulo respecto a la visión del robot, para poder ofrecer una mejor localización.

En el pasillo de la parte central izquierda de la zona de oficinas, el robot solo visualiza dos balizas, por lo que al tener que avanzar una larga distancia se pone en compromiso su localización.

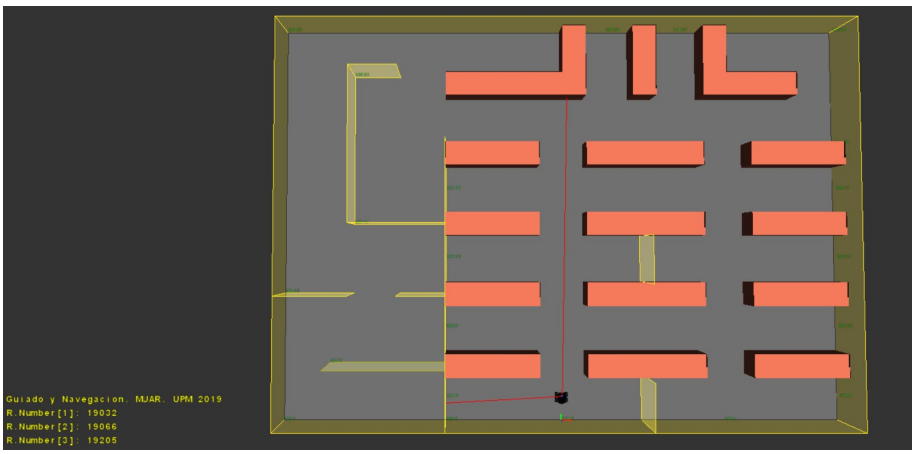


Figura 1 Planta del entorno.

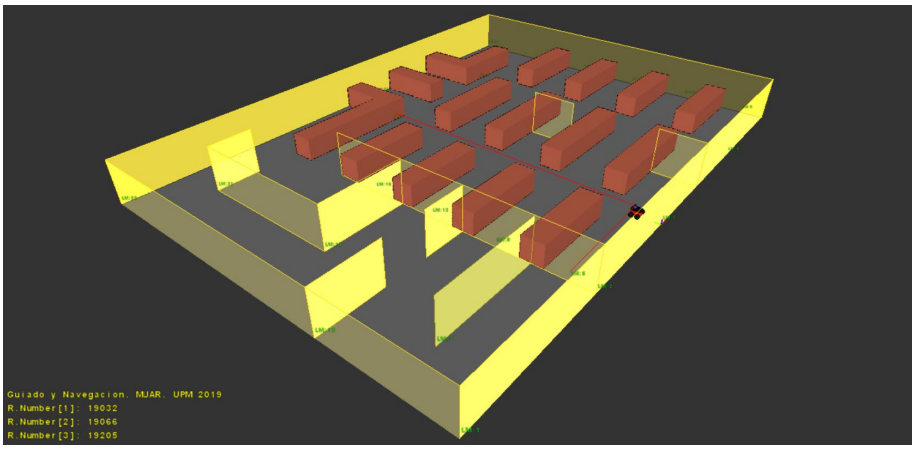


Figura 2 Vista en perspectiva del entorno.

4. ROBOT

Para el Desarrollo de este proyecto se pensó en los distintos sensores disponibles, así como distintas configuraciones. Sin embargo, los diseños propuestos eran casi iguales a un robot ya implementado en el programa. Por esta razón, y tras una larga discusión sobre las ventajas de otra configuración, se decidió usar un robot predefinido. Se trata del robot 'Marvin'.

Este robot posee un sensor láser en su parte superior, con un rango de detección de 270 grados. Adicionalmente cuenta con tres sensores de ultrasonidos, todos situados en la parte frontal apuntando dos de ellos a las esquinas frontales. Esto permite controlar una posible colisión tanto en avance positivo con en cualquier rotación. Además, tiene una odometría basada en giro diferencial, permitiéndole rotar en un mismo punto, lo cual facilita su manejo en gran medida.

Como curiosidad, ya que uno de los modelos propuestos tenía el nombre de 'Paco', decidimos renombrar a Marvin para acompañar al equipo durante el largo desarrollo de los algoritmos.

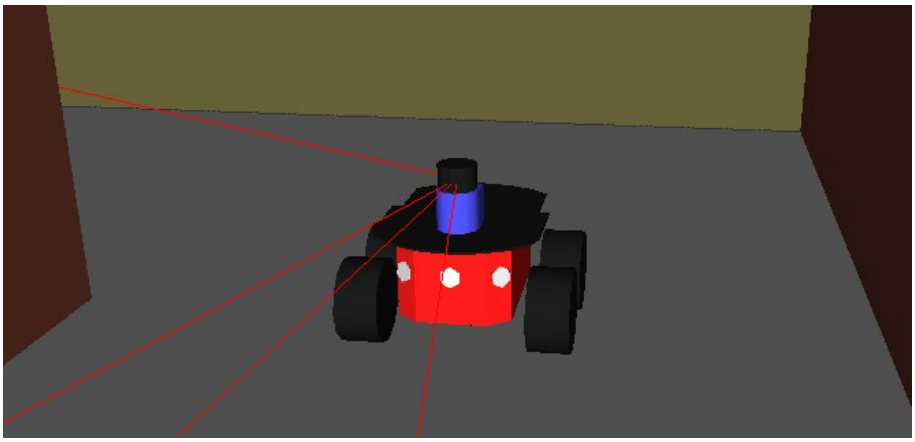


Figura 3 Robot utilizado. Paco.

5. GRÁFICA

Durante la ejecución del algoritmo y la simulación aparecerá una gráfica que indica el comportamiento y resultado de las siguientes etapas. Este el desglose (leyenda) de los distintos elementos según orden de aparición:

Durante la ejecución del algoritmo planificador

- -Paredes y estanterías en color gris: mapa del entorno.
- -Puntos rojos: Balizas láser.
- -Círculo azul: Nodo origen del planificador.
- -Círculo rojo: Nodo destino del planificador.
- -Rectas negras: Árboles de búsqueda.
- -Recta azul: Camino solución.
- -Recta roja: Camino solución suavizado y optimizado.

Durante la simulación de Apolo

- -Círculo verde: Punto de origen.
- -Puntos grises: Punto objetivo del robot.
- -Puntos negros: Posición real del robot.
- -Puntos magenta: Posición dada por el localizador.
- -Línea magenta: Orientación dada por el localizador.
- -Círculo magenta: Punto de destino alcanzado.

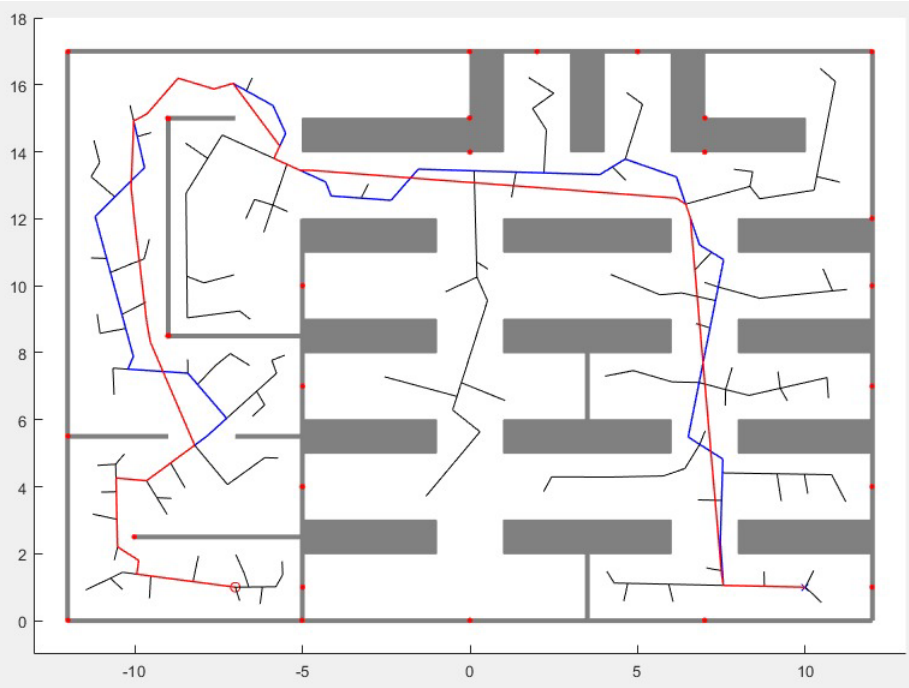


Figura 4 Algoritmo RRT-Connect finalizado (azul) junto a la optimización de ruta (rojo).

Figura 5 Simulación acabada.

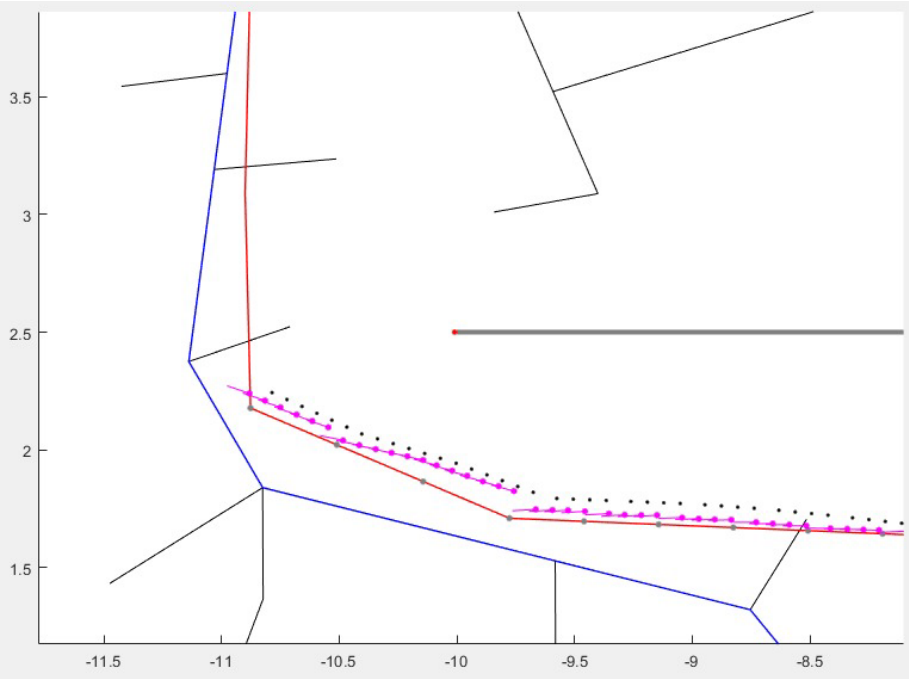


Figura 6 Ampliación en detalle de la gráfica durante la ejecución de la simulación.

6. MANUAL DE EJECUCIÓN DE NUESTRA IMPLEMENTACIÓN

Para la correcta ejecución de nuestro programa deben seguirse los siguientes pasos:

1. Los cambios se harán mediante la función 'configuración.m'. En ella podemos variar el algoritmo utilizado (RRT o RRT-Connect), los nodos inicial (con su orientación) y final y las variables relativas al entorno de Apolo (nombre del robot, mundo y sensores).
2. Apolo debe de estar activo con el entorno (mapa.xml) cargado.
3. Las carpetas de funciones de Apolo de Matlab y las funciones de nuestra implementación añadidas al 'Path' de Matlab. En caso contrario aparecerá un cuadro de diálogo recordándolo.
4. Ejecutar el script 'main.m'. Esto arrancará el algoritmo de planificación y seguidamente la simulación, mostrando las gráficas correspondientes.

En el cuadro de comandos de Matlab irán apareciendo mensajes de orientación para que el usuario conozca el estado de ejecución de todo el programa. Debe notarse que existen algunas pausas estratégicamente colocadas para garantizar la fluidez y comunicación entre ambos programas. La más notoria ocurre antes de empezar a simular el movimiento del robot en Apolo.

7. CALIBRACIÓN

La calibración de los sensores es importante para saber cuán fiable es la información que el robot recibe de estos, ya que parte de esta información es la que usará para localizarse en el espacio y corregir su trayectoria.

Por esta razón se ha desarrollado un script que el usuario puede ejecutar para conocer la varianza de los resultados obtenidos para cada uno de los distintos sensores que el robot hace uso, así como el error de su media.

Antes de la ejecución el usuario debe especificar nombre del robot, el nombre del mundo y el tipo de calibración que desea realizar mediante la asignación del valor correspondiente a la variable tipo, pudiendo escoger entre las siguientes:

1. Calibración de la odometría lineal
2. Calibración de la odometría angular
3. Calibración de los sensores láser
4. Calibración de los sensores de ultrasonido

7.1. Pruebas de calibración (calibracion.m)

Calibración de la odometría lineal

Inicialmente se posiciona el robot en la localización inicial y una vez correctamente posicionado, mediante un bucle, se le ordena repetidas veces avanzar durante un segundo con una velocidad lineal de $0{,}5\ unidades/seg$. En cada una de las iteraciones del bucle, antes de ejecutar el movimiento, se reinicia la odometría para obtener los resultados de cada uno de los distintos avances por separado. Una vez almacenados los resultados de cada una de las iteraciones se puede calcular la varianza total y la media obtenida, que permite obtener su error al restarle la distancia ideal que debería haber recorrido el robot tras el movimiento.

Calibración de la odometría angular

El procedimiento es muy similar al procedimiento de la calibración lineal; tras colocar el robot en la posición inicial, de nuevo con un bucle, se le ordena al robot realizar múltiples movimientos de giro con una velocidad angular de $1\ rad/seg$. Antes de realizar cada uno de los movimientos se reinicia la odometría con tal de obtener resultados individuales para cada uno de los giros. Al igual que en el módulo anterior se obtiene la varianza total, así como la media, a la que se le resta el giro que debería haber recorrido idealmente.

Calibración del sensor láser

Para la calibración del sensor láser se posiciona al robot a diferentes distancias de la baliza indicada y se realiza la toma de medidas mediante dos bucles. Al terminar cada una de las medidas para una determinada distancia se aleja el robot y se vuelven a realizar las medidas para la nueva distancia. Este proceso se repite para cada una de las distancias fijadas.

Con todos los resultados almacenados se calcula lo siguiente:

- La varianza obtenida para cada distancia
- El valor medio obtenido para cada distancia
- El error del valor medio para cada distancia

Finalmente se muestran dos gráficos, el primero muestra la varianza en función de la distancia junto al valor medio de las varianzas; el segundo muestra el error de la medida en función de la distancia junto al valor medio de los errores.

Calibración del sensor de ultrasonido

En el último módulo se vuelven a realizar dos bucles, en el primero se coloca al robot a diferentes distancias de una pared y en el segundo se toman las diferentes medidas. Tras finalizar la ejecución se muestra el valor de la varianza para cada una de las distancias, así como el valor medio.

Por último, al igual que en el módulo anterior, se muestra un gráfico con la varianza en función de la distancia con su valor medio y en el segundo se muestra el error de la medida frente a la distancia junto con su valor medio.

Para los dos primeros módulos se toman medidas para desplazamiento y giro fijo debido a que durante el trayecto el robot se desplaza linealmente y gira distancias muy pequeñas.

Para los dos últimos se toman medidas a varias distancias de la baliza y de la pared porque el robot puede encontrarse más o menos alejado de dichos elementos, siendo de suma importancia estudiar el comportamiento en esos casos, y su evolución en función de la distancia.

7.2. Resultados de la calibración

Test de calibración de la odometría lineal

Las pruebas realizadas consisten en ordenarle al robot la realización de un movimiento lineal con velocidad $0.5\ unidades/seg$ durante un segundo. Tras repetir este movimiento mil veces se obtienen los siguientes resultados:

- Varianza: $3.046252e^{-04}$
- Error de la media: $3.913249e^{-04}$

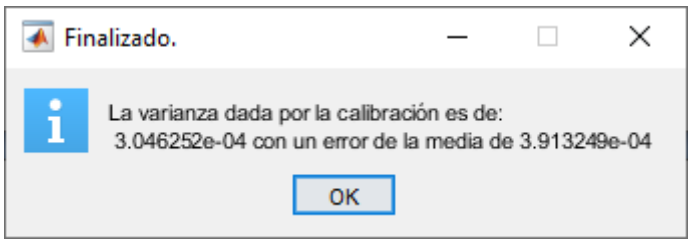


Figura 7 Resultado de la calibración odométrica lineal.

Tal como se puede apreciar, la variación de resultados y el error en la medida son prácticamente despreciables.

Test de calibración de la odometría angular

Las pruebas realizadas consisten en ordenarle al robot la realización de un movimiento angular con velocidad $1\,rad/seg$ durante un segundo. Tras repetir este movimiento mil veces se obtienen los siguientes resultados:

- Varianza: $2.933419e^{-04}$
- Error de la media: $9.763574e^{-05}$

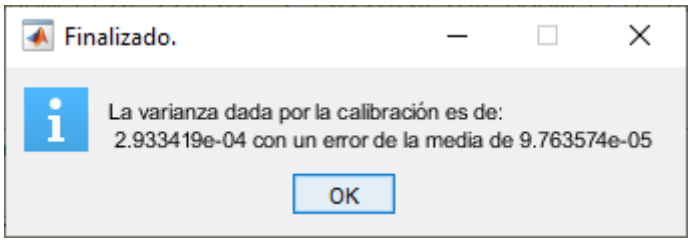


Figura 8 Resultado de la calibración odométrica angular.

Tal como se puede ver, la variación de resultados y el error en la medida son prácticamente despreciables.

Test de calibración del sensor láser

Las pruebas realizadas consisten en situar el robot a una unidad de distancia de la baliza y tomar mil medidas de la distancia que calcula el sensor. Este proceso se repite hasta alcanzar una

distancia de diez unidades con incrementos de media unidad. Una vez terminadas las 19 medidas se obtienen los siguientes resultados:

- Varianza: $1.462576e^{-04}$
- Error de la media: $-4.574645e^{-03}$

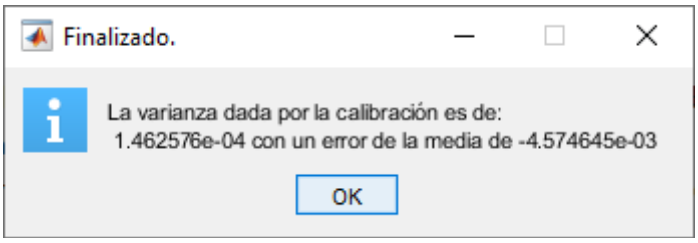


Figura 9 Resultado de la calibración del sensor láser.

Estos resultados muestran que tanto la varianza total como el error en la medida total son prácticamente despreciables.

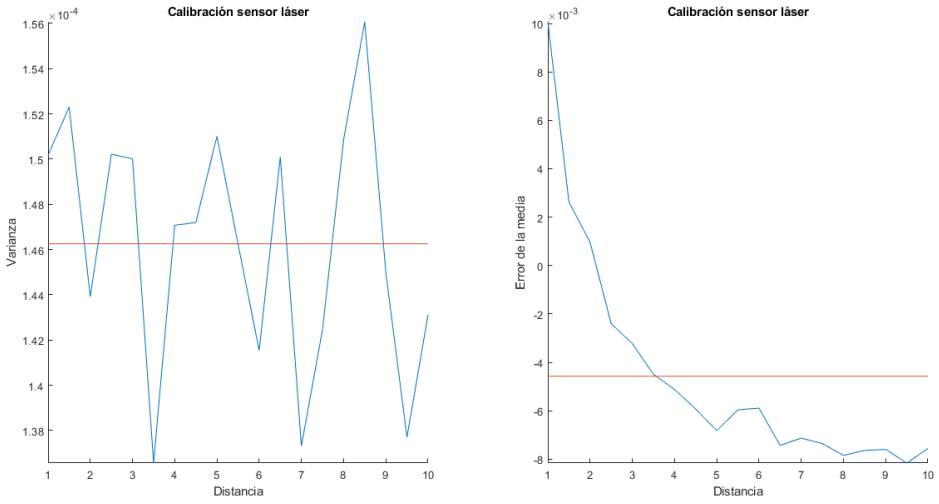


Figura 10 Gráficas en función de la distancia hasta el láser.

Gracias a la primera gráfica de la imagen anterior se puede afirmar que no existe una relación entre la varianza de las medidas y la distancia a la cual se están tomando los valores. En la segunda gráfica se puede ver como a menor distancia a la baliza mayor es el error positivo de la medida, por otra parte, conforme va aumentando la distancia, disminuye el error hasta lo que parece una estabilización en valores negativos a partir de las diez unidades de longitud

.

Test de calibración del sensor de ultrasonido

Las pruebas realizadas consisten en situar el robot a media unidad de distancia de la pared y tomar mil medidas de la distancia que calcula el sensor. Este proceso se repite hasta alcanzar una distancia de tres unidades con incrementos de un cuarto de unidad. Una vez terminadas las 19 medidas se obtienen los siguientes resultados:

- Varianza: $9.922153e^{-28}$
- Error de la media: $-1.370765e^{-02}$

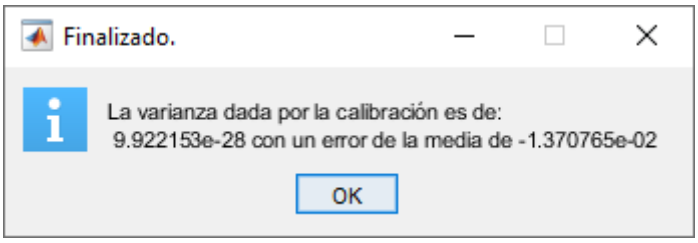


Figura 11 Resultado de la calibración de los sensores ultrasónicos.

Tras ver estos resultados se pueden considerar despreciables.

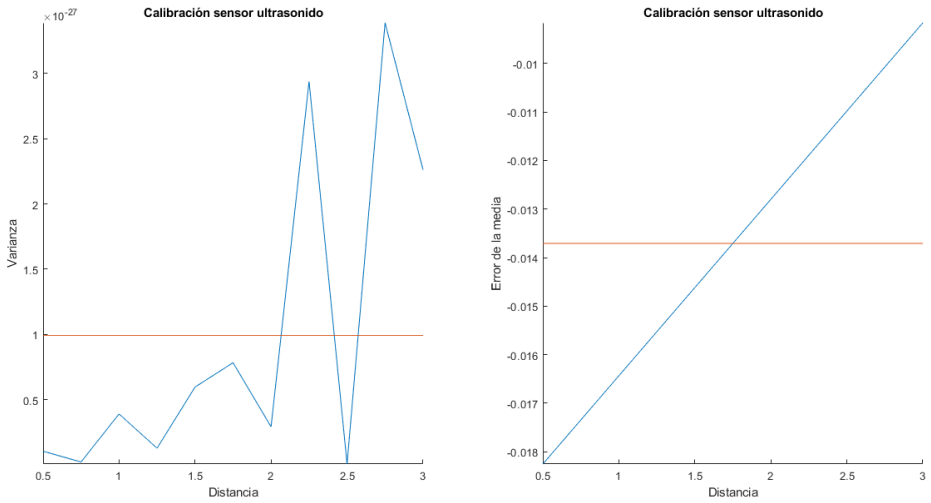


Figura 12 Gráficas en función de la distancia hasta el láser.

Tras ver el primer gráfico en la imagen anterior se puede ver que la varianza de la medida va aumentando en función de la distancia entre el robot y la pared. En el segundo gráfico se puede

comprobar que conforme la distancia va en aumento el error se va reduciendo de manera proporcionalmente inversa.

Todos los valores obtenidos en los experimentos anteriores se replican de manera muy similar al repetirlos, por lo que podemos usarlos con seguridad.

8. PLANIFICACIÓN

Para la implementación de un planificador para este proyecto, se ha optado por utilizar uno basado en RRT (de las siglas 'Rapidly-Exploring Random Tree'). Este permite extenderse de forma rápida ya que está basado en la creación de nodos aleatorios, lo que ahorra bastante computación.

8.1. Implementación del RRT

Inicialización

Se definen un punto origen y un punto destino, así como un valor de incremento entre puntos de aproximación en el árbol.

Se define también un número de iteraciones de intento de conexión con el punto destino. Cada vez que el bucle general alcanza dichas iteraciones, el algoritmo intenta conectarse con el punto de destino en vez de con un punto aleatorio. Tras cada intento se reestablece un contador interno que vuelve a contar el número de iteraciones hasta alcanzar la siguiente comprobación.

Para la creación de los árboles se ha optado por trabajar con estructuras. Un árbol estructura está compuesto por tres campos que corresponden a los datos:

- Coordenada x
- Coordenada y
- Índice de la posición del padre del punto (x, y), es decir, el punto que le precede. De esta forma un punto siempre conocerá el punto desde el cual se llega hasta él en forma de su índice de colocación en la estructura del árbol.

El árbol se inicializa con el nodo origen, asociando un índice 0 (inexistente en MATLAB), que será posteriormente utilizado para identificar el fin del 'backtracking' (caso base). Una vez creado el árbol, se comprueba que el punto origen sea un punto válido, obteniendo como salida un valor 1 en la función choque. De la misma forma, se comprueba que el punto destino sea válido. En caso contrario, se escribirá en pantalla que el punto dado no es válido y se terminará la ejecución.

De ser válidos los puntos origen y destino, se entra entonces en un bucle 'for', que tiene una extensión máxima de iteraciones configurada por el usuario en la variable 'MaxIterations'.

El bucle del algoritmo

Una vez creado el árbol, se generan puntos aleatorios en un área rectangular predefinida por el usuario. El árbol calcula su punto más cercano al punto aleatorio generado mediante la función 'nearest.m'. Una vez se tengan ambos puntos, se calcula la distancia entre ellos y se utiliza la función 'linspace' de MATLAB para discretizar una línea que los una a ambos.

Estos puntos son candidatos para añadirse al árbol. Una vez se tienen los puntos intermedios candidatos se entra en un bucle 'for' que comprueba mediante la función correspondiente si chocan o no con algún objeto del mapa.

- En el caso de que no choque, se agrega el punto intermedio candidato a un vector temporal que será utilizado para graficarlo más adelante y se añade al árbol. Si este es el primer punto candidato a ser añadido al árbol, su padre será el de unión con el mismo y, en caso contrario, punto intermedio anterior añadido.
- En el caso de que se detecte que un punto intermedio choca, se deja de comprobar los puntos intermedios restantes.

Una vez fuera del bucle que recorre los puntos intermedios, se comprueba si el último punto válido añadido al árbol se corresponde con el punto destino. En caso afirmativo, la ejecución sale del bucle general y se prosigue con la obtención del vector de soluciones. Si no, itera el bucle y grafica los puntos nuevos del árbol para tener su representación gráfica.

Cuando la ejecución encuentra la solución y sale del bucle general, se procede a la obtención del vector de puntos solución. Para ello se pasa como valor todo el árbol y el punto de origen a la función 'backtracking_RRT.m'. Esta devuelve una matriz de puntos solución ordenados.

Finalmente, se dibuja encima del árbol la solución final del algoritmo RRT en un color distinto para su diferenciación.

Etapa de 'smoothing'

Dado a que la matriz de puntos soluciones obtenida por el planificador RRT posee giros bruscos y no resulta del todo directa, se ha decidido implementar una función que trate de suavizar esa trayectoria solución. Para ello se ha definido la función 'smoothing.m', que recibiendo una matriz de puntos (x, y) y un valor de incremento, realiza un número determinado de iteraciones eligiendo dos puntos aleatorios de la matriz de entrada e intenta unirlos creando puntos intermedios entre ellos según el incremento en línea recta. Si en ese intento de unión, algún punto intermedio creado choca, se aborta el intento de unión y se pasa a la siguiente iteración. En el caso de que no choque ningún punto en el intento de unión, se reemplazarán los puntos intermedios de la matriz por los nuevos creados.

Como salida, se obtiene una ruta solución mucho más suave y optimizada que elimina picos y retrocesos en la solución. Esto ayuda muchísimo cuando el robot ha de recorrer dicha ruta, ya que resulta una trayectoria más dinámica y directa.

8.2. RRT-Connect

Justificación de uso

Hay casos en los que el punto de destino se sitúa en encajonamientos en los que solo hay una pequeña entrada. Esta situación hace que sean necesarias muchas iteraciones para que el árbol de búsqueda crezca lo suficiente como para acercar puntos a esa pequeña entrada. Dada la

metodología del algoritmo RRT, se depende en gran medida de la aleatoriedad y de la espera de que el árbol crezca en la dirección deseada.

Una posible mejora ante dicha situación es la implementación de un algoritmo RRT-Connect, que básicamente usa dos árboles, uno con origen en el punto de origen y otro con origen en el punto destino. Esto hace la búsqueda de ruta mucho más dinámica y facilita en gran medida la resolución de encajonamientos. Es por ello por lo que el equipo decidió su implementación a partir del algoritmo RRT ya desarrollado.

Implementación del RRT-Connect

El algoritmo tipo RRT-Connect implementado sigue la siguiente metodología, ya que hemos considerado que es la que mejor nos funciona:

Los primeros pasos los realiza igual que la función del RRT, salvo que esta vez inicializa un nuevo árbol (denominando árbol 2 de aquí en adelante), cuyo origen es el punto destino. Una vez inicializados los dos árboles, se entra entonces en el bucle general, donde se puede diferenciar claramente cuatro partes.

1. Se crea un punto aleatorio. El árbol 1 intenta unirse a este según la metodología del RRT implementado anteriormente.
2. Una vez ha finalizado la aproximación del árbol 1 al punto aleatorio, este puede haber conseguido unirlo al árbol sin chocar en el trayecto de unión, o puede haber chocado antes. En cualquier caso, último punto del árbol 1 será el más cercano al punto aleatorio propuesto. Es entonces cuando se propone al árbol 2 el último punto del árbol 1 como punto objetivo. Esto hace que el árbol 2 se intente conectar con el árbol 1. En el caso de conseguirlo, se terminaría la ejecución del bucle general y se pasaría a obtener la matriz de puntos solución.
3. Tras no conseguir unirse con el árbol 1, el árbol 2 se intentará conectar con un nuevo punto aleatorio.
4. Ahora es el turno del árbol 1, el cual intentará conectarse con el último punto añadido al árbol 2. En el caso de conseguirlo, se terminaría la ejecución del bucle general y se pasaría a obtener la matriz de puntos solución.

Una vez los dos árboles se encuentran, se ejecuta la función ‘backtracking_RRTCONNECT.m’ correspondiente para cada árbol. A la matriz de puntos solución obtenida del árbol 1 se le da la vuelta, y se concatena con la matriz de puntos solución del árbol 2. Por último, la matriz de puntos solución se devuelve como parámetro.

9. LOCALIZACIÓN

9.1. Funcionamiento general

Nuestro algoritmo de localización se basa en el modelo del Filtro Extendido de Kalman, particularizado para el uso de medidas de distancia, con el fin de cambiar la implementación cedida por el profesor (basada en ángulos). El algoritmo sigue los siguientes pasos.

Inicialización

Se establecen el número de balizas (que determina las dimensiones de las matrices utilizadas) y las matrices de varianza del proceso (odometría) y medida (de los sensores láser), que son resultado del proceso de calibración.

Etapa de predicción

Utilizando las medidas de odometría y los valores del estado anterior, se estima, teniendo en cuenta los sistemas de referencia de los datos, los valores del estado actual. Siendo el sistema de referencia inercial el que proporciona las coordenadas del mapa y de las gráficas y el sistema de referencia de la odometría el sistema propio del robot (avance en la dirección del eje x).

A continuación, se predice la matriz P_k utilizando las matrices jacobianas de la expresión anterior, dada en función de senos y cosenos.

Luego, se predice, teniendo los valores de las posiciones de las balizas, todas las distancias desde el estado estimado a todas las balizas. En etapas posteriores se seleccionarán solo las predicciones de la medida necesarias. Por último, se establece su matriz asociada dada la matriz jacobiana de la expresión de cálculo de distancia.

$$X = \begin{pmatrix} x + u_x \cdot \cos(\theta) \\ y + u_x \cdot \sin(\theta) \\ \theta + u_\theta \end{pmatrix}$$

$$Jacobian(X, [x\ y\ \theta]) = \begin{pmatrix} 1 & 0 & -u_x + \sin(\theta) \\ 0 & 1 & u_x + \cos(\theta) \\ 0 & 1 & 0 \end{pmatrix}$$

$$Jacobian\big(X, [u_x\ u_y\ u_\theta]\big) = \begin{pmatrix} \cos(\theta) & 0 & 0 \\ \sin(\theta) & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Etapa de observación

Se obtienen las medidas reales de las balizas visibles por el sensor láser con el comando 'apoloGetLaserLandMarks' y se colocan en un vector de medida de dimensión el número total de balizas, en la posición de índice correspondiente con su identificador. Esta implementación sirve para que la dimensión del vector de medida observada sea congruente con el de medida predicha.

$$Zk\_(i,1) = \sqrt{\left(ty - Xk_y\right)^2 + (tx - Xk_x)^2}$$

$$Hk(i,:) = \left(-\frac{2 \cdot tx - 2 \cdot Xk_x}{2 \cdot \left((tx - Xk_x)^2 + \left(ty - Xk_y\right)^2\right)^{\frac{1}{2}}} \quad -\frac{2 \cdot ty - 2 \cdot Xk_y}{2 \cdot \left((tx - Xk_x)^2 + \left(ty - Xk_y\right)^2\right)^{\frac{1}{2}}} \quad 0 \right)$$

Etapa de comparación

Se restan los vectores de medida observada y medida predicha para obtener un vector de innovación de la medida. A este hay que lleva a 0 los valores de las posiciones cuyas balizas no hayan sido observadas en la etapa de observación. Luego, se calcula la varianza de la innovación y la ganancia de Kalman, que son necesarias para la etapa final.

Etapa de corrección

Aquí se calculan los valores finales y de salida de la función del estado y su varianza, dadas las expresiones de su formulación. Estas ajustan la importancia relativa entre la etapa de predicción y observación y su contribución a la nueva estimación del estado. Por último, ajusta los valores angulares para mantenerlos en un intervalo $[-\pi, \pi]$.

Al finalizar la función se reinicia la odometría del robot preparándolo para la siguiente iteración.

9.2. Elección de los parámetros iniciales del estado

Se parte de la premisa de que el robot ha sido colocado en una posición conocida, pero con un posible error. Esto se introducirá artificialmente en el algoritmo desviando la posición inicial (x, y, θ) con la que entra por primera vez en el bucle. Haciendo una estimación del orden del error de la colocación, que en este caso se presupone en las décimas de metro, se establece también el valor inicial de la matriz de varianzas y covarianzas del estado.

9.3. Estudio de los parámetros iniciales de localización

Tras variar los parámetros de inicio del algoritmo de localización (x, y, θ, P) de distintas maneras, aplicando un error tanto en longitud como en ángulo, se han obtenido las siguientes conclusiones.

- Los valores de la matriz de varianzas y covarianzas, variando en intervalos entre 0.01 y 1, parecen influir poco en el comportamiento del algoritmo.
- Los errores de ángulo inicial producen grandes desviaciones en el comportamiento del algoritmo y en las estimaciones del localizador. Una vez se excede de $\pi/3$, el resultado es muy malo.
- El localizador es más robusto a cambios de posición en (x, y), soportando desviaciones del orden de la unidad.

En la imagen se puede observar la desviación de la estimación causada por un error en la medida de la unidad en ambos ejes. Ruta magenta en la esquina inferior izquierda. Al final, consigue localizarse correctamente.

Figura 13 Comportamiento con desviación inicial incrementada.

9.4. Impacto de las matrices de varianza de la medida del láser y la odometría

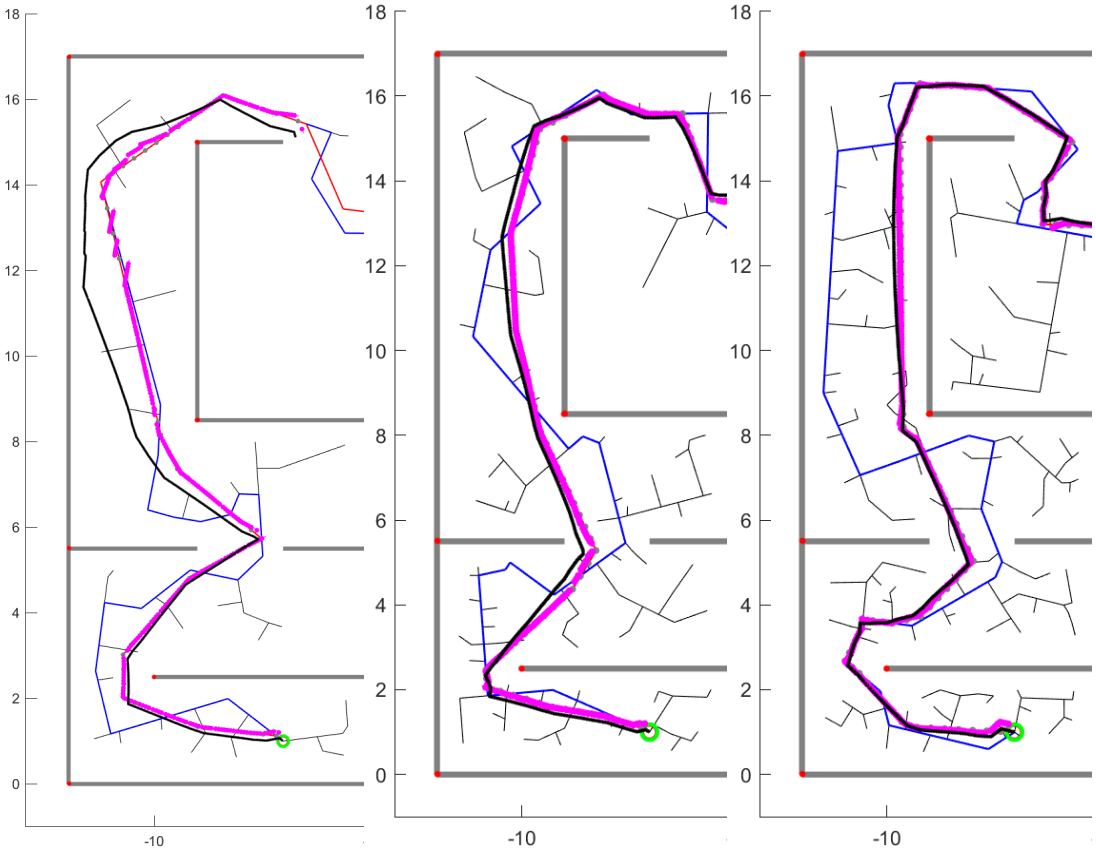


Figura 14 R = R(real)·100 [izquierda]

Figura 15 Qlin = Qlin(real)·100 [centro]

Figura 16 Qrot = Qrot(real)·100 [derecha]

Tras variar los parámetros de calibración individualmente con un multiplicador de 100 se pudo observar la importancia relativa de cada uno de ellos para el conjunto del sistema, siendo la calibración del sensor láser la más importante y crítica para el comportamiento correcto del robot. Es por ello por lo que en las zonas el mapa donde se observan menos balizas, la localización del robot tiende a acumular un error mayor, corrigiéndose cuando existen más balizas observables.

En otras palabras, el algoritmo de localización tiene una mayor dependencia de las medidas con las balizas y funciona mejor cuantas más tiene en el campo de visión del sensor láser.

10. CONTROL Y REACTIVIDAD

10.1. Funcionamiento general

Inicialización

Una vez desarrollada la localización, se puede proceder al diseño del control.

Primeramente, se da valor a un primer estado para la localización, el cual se da erróneo a propósito, encontrándose desviado respecto a la posición verdadera para que el apartado de localización lo corrija. Se inicializa también la matriz de varianzas y covarianzas del estado en función del grado de error inicial estimado y se marca en la gráfica el punto inicial, indicado por un círculo verde.

El bucle

El controlador trata de seguir las posiciones que da el vector solución de puntos, así que se entra en un bucle 'while' que recorre toda esta vector de puntos, buscando el siguiente punto a cada iteración hasta que se finaliza el vector solución. Esto se consigue estableciendo un punto objetivo y llevando a cabo las siguientes acciones:

- Orientación, alineándose con el punto.
- Avance, basado en un cálculo de la distancia a recorrer.

Reactividad

Una vez conocido el punto objetivo se comienza comprobando la distancia de los sensores de ultrasonidos. Eso detecta si el robot se inicia en una posición comprometida. Se trabaja con tres sensores de ultrasonidos, una en la esquina frontal izquierda del robot, otro en el centro del frontal y el ultimo se encuentra en la esquina frontal derecha del robot. Esta configuración de sensores ofrece gran control sobre los obstáculos en la dirección de avance del robot, así como obstáculos que se aproximen por el lateral.

Si algún sensor de ultrasonidos detecta una distancia menor a 0.2 (parámetro obtenido de forma experimental) se entra en una etapa de reactividad y se llama a la función 'reactivo.m'.

El controlador entra en un bucle 'while' del que no sale hasta que la distancia detectada por cada sensor de ultrasonidos sea mayor de 0.35 dando una tolerancia entre el umbral de entrada a la reactividad y de salida de su bucle.

Se comprueban entonces tres posibilidades:

- Sensor derecho detecta obstáculo más próximo que el sensor izquierdo. En este caso se establece un sentido de giro antihorario.

- Sensor izquierdo detecta obstáculo más próximo que el sensor derecho. En este caso se establece un sentido de giro horario.
- Ambos sensores detectan un obstáculo a la misma distancia. En este caso se hace rotar al robot $0.6\ radianes$. Con este giro se pretende que los dos sensores dejen de detectar un objeto a la misma distancia, entrando en un ciclo que lo inmoviliza.

Se girará en función de la condición establecida hasta que se consigue cumplir la condición de abandono del bucle.

Una vez resuelto el peligro de obstáculos cercanos, se llama a la función 'Localizacion.m' la cual actualiza el estado del robot (posición y orientación) y la matriz de varianzas y covarianzas del estado.

Debido a que la componente reactiva ha descolocado el robot, este trata de encontrar, desde su posición recién obtenida, el punto más cercano a la matriz de puntos solución. Entonces se actualiza el punto objetivo, guardándose el índice y fijándolo al bucle, continuando el de manera normal el bucle principal.

Condición de orientación

Se continúa calculando el ángulo al que se encuentra el punto objetivo, respecto a la posición actual, y se redondea a las centenas. Nuevamente se actualiza el estado mediante la función 'localizacion.m', del cual se obtiene el ángulo actual del robot y también se redondea a las decenas. Estos son los valores que se van a comparar para establecer los comandos de giro del robot. Trabajaremos con ángulos contenidos en el rango $[-\pi, \pi]$.

En el caso de que ambos ángulos sean distintos, se procede a tratar de igualar el ángulo del robot al ángulo objetivo mediante un movimiento de rotación. Para determinar en qué sentido ha de girar el robot para que el robot realice siempre la menor rotación, se lleva a cabo la siguiente comprobación de casos.

- Si el ángulo objetivo tiene signo distinto al ángulo actual del robot y el ángulo actual del robot no es cero.
  - Si el ángulo objetivo es negativo y menor que $-\pi/2$. El robot gira en sentido antihorario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales.
  - Si el ángulo objetivo es negativo y mayor que $-\pi/2$. El robot gira en sentido horario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales.
  - Si el ángulo objetivo es positivo y menor que $\pi/2$. El robot gira en sentido antihorario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales.
  - Si el ángulo objetivo es positivo y mayor que $\pi/2$. El robot gira en sentido horario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales

- Si el ángulo objetivo tiene igual signo al ángulo actual del robot.
    - Si el ángulo objetivo es mayor que el ángulo actual del robot. El robot gira en sentido antihorario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales.
    - Si el ángulo objetivo es menor que el ángulo actual del robot. El robot gira en sentido horario mientras va actualizando su estado con la función de localización hasta que ambos ángulos, actual y objetivo, sean iguales.

Una vez orientado correctamente el robot hacia el objetivo, se grafica el punto objetivo con un pequeño círculo gris sobre la línea solución. Esto ayuda a entender visualmente cual va siendo el siguiente objetivo del robot.

Condición de avance

Ahora llega el turno de realizar una aproximación lineal al punto objetivo. Esta se realiza calculando la distancia desde la posición actual del robot hasta el punto objetivo. Para ello se ha vuelto a actualizar previamente la localización del robot mediante la función 'localización.m'.

Una vez conocida la distancia, y establecida al inicio la velocidad lineal, se conoce los incrementos de distancia que realiza el comando de movimiento del robot en la componente lineal. Se entra en un bucle 'for', que realiza tantos movimientos lineales como son necesarios hasta alcanzar la distancia anteriormente calculada. En cada pequeño avance, el controlador vuelve a actualizar los sensores de ultrasonidos, para en caso de detectar algún obstáculo, volver a entrar en el procedimiento reactivo mencionado anteriormente.

La posición del robot se va actualizando en cada pequeño incremento mediante la función de localización y se va graficando, junto con la posición real del robot en el simulador obtenida mediante la función 'apoloGetLocationMRobot'. Mediante la representación visual de la posición real del robot y la posición proporcionada por la función de localización, el usuario puede apreciar la exactitud del algoritmo y del procedimiento de este.

Finalmente, cuando el robot alcanza la posición final del vector de puntos solución, sale del bucle del controlador y grafica un círculo de color magenta sobre el punto final.

11. BIBLIOGRAFÍA

Para los planificadores:

[1] Hernando, M. (s.f.). Tema 10. Planificación basada en muestreo. Madrid, España: UPM.

Idea de la función 'smoothing':

[2] Touestzky, D. S., & Tira-Thompson, E. (2012). Path Planning. Recuperado de https://pdfs.semanticscholar.org/0731/8235562d6e3e06461676b55e1f2ff3d2f68a.pdf