UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

MÁSTER EN AUTOMÁTICA Y ROBÓTICA

GUIADO Y NAVEGACIÓN DE ROBOTS

TRABAJO FIN DE ASIGNATURA

Autores:

Álvaro Benito Oliva (M20159)

Germán Andrés Di Fonzo Caturegli (M20037)

Juan José Jurado Camino (M20039)

Tutores:

Fernando Matía

Miguel Hernando

Paloma de la Puente

Madrid, 17 de enero de 2021

TABLA DE CONTENIDO

1 OBJETIVO DEL TRABAJO Y REPARTO DE ROLES

El principal objetivo que se busca con este trabajo es familiarizarse con los temas más importantes que surgen en el momento de programar la navegación de un robot en presencia de incertidumbre:

- Sistema de locomoción del robot móvil (odometría).
- Elección de los sensores exteroceptivos.
- Algoritmos para la estimación del estado del robot (localización).
- Algoritmos de control de movimientos.
- Algoritmos de planificación de trayectorias.

Para ello, se decide escoger una aplicación de navegación de un robot móvil y se implementará en simulación mediante MATLAB y Apolo.

El reparto de roles utilizado para la consecución de este trabajo ha sido el siguiente:

- Germán Di Fonzo: mapa del entorno, calibración y localización.
- Juan José Jurado: control y localización.
- Álvaro Benito: planificación de trayectorias y localización.

2 PLANTEAMIENTO DEL PROBLEMA

La aplicación que se ha decidido escoger para este trabajo consiste en un robot encargado de limpiar el suelo de un restaurante o cafetería con dos comedores. Este robot podría resultar muy útil cuando los clientes ya se han ido y el restaurante está a punto de cerrar. De esta manera, mientras los camareros recogen y limpian las mesas, el robot podría ir limpiando el suelo.

Para poder simular en Apolo el movimiento del robot a lo largo de los comedores del restaurante, se ha programado un mapa del entorno con el lenguaje XML en un archivo denominado *entornoRestaurante.xml*. Este mapa es el que se muestra en la *Figura 1* y se caracteriza por ser una sala cuadrada de 16x16 metros con 7 mesas y un muro en medio que separa dos comedores. La mesa de mayor longitud se corresponde con la mesa donde se atiende a los clientes, mientras que las otras mesas se utilizan para que los clientes puedan sentarse y comer.

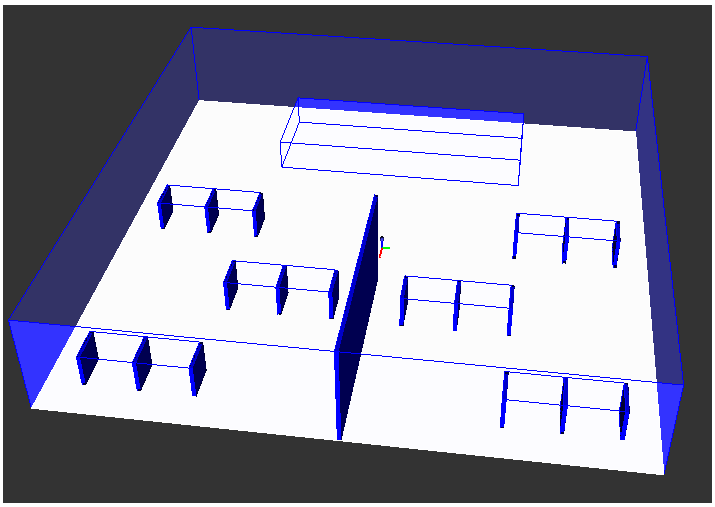

Figura 1. Mapa del entorno visualizado en Apolo.

3 SISTEMA DE LOCOMOCIÓN

El robot con el se trabaja en este proyecto es un Pioneer3AT. Este robot se caracterizar por tener un sistema de locomoción tipo diferencial.

En el simulador facilitado ya viene contenido el modelo cinemático del sistema de locomoción de tipo diferencial. Por lo tanto, para la calibración de este sistema se ha seguido el siguiente procedimiento:

1) Se coloca el robot móvil en el origen de coordenadas del mundo, el cual coincide con el centro de la sala cuadrada. La orientación inicial del robot es de 0 rad.

2) Se reinicia la odometría a un valor de [0 0 0] con la función *apoloResetOdometry()*. De esta manera, al colocar el robot en el origen de coordenadas del mundo, el vector *[x y θ]* proporcionado por la función *apoloGetOdometry()* coincidirá con la estimación de la pose del robot según la odometría.

3) Se definen los parámetros del movimiento del robot para que describa una elipse. Estos parámetros se han elegido de manera que sean muy parecidos a los valores con los que se trabajará en los algoritmos de localización y control del robot.

   - Velocidad lineal: 0.5 m/s.
   - Velocidad angular: 0.3 rad/s
   - Tiempo del movimiento: 0.1 s
   - Número de iteraciones: 1000

4) Se calcula la pose real del robot con la función *apoloGetLocationMRobot()* y la pose estimada según la odometría con *apoloGetOdometry()* en cada iteración. Estos valores de pose real y estimada se van almacenando para calcular posteriormente el error absoluto cometido en cada variable de estado. En la *Figura 2* se puede observar la trayectoria que describe el robot según la odometría (elipses de color azul) y la trayectoria real (elipse de color roja). Como se puede observar, la incertidumbre de la odometría es bastante notoria.

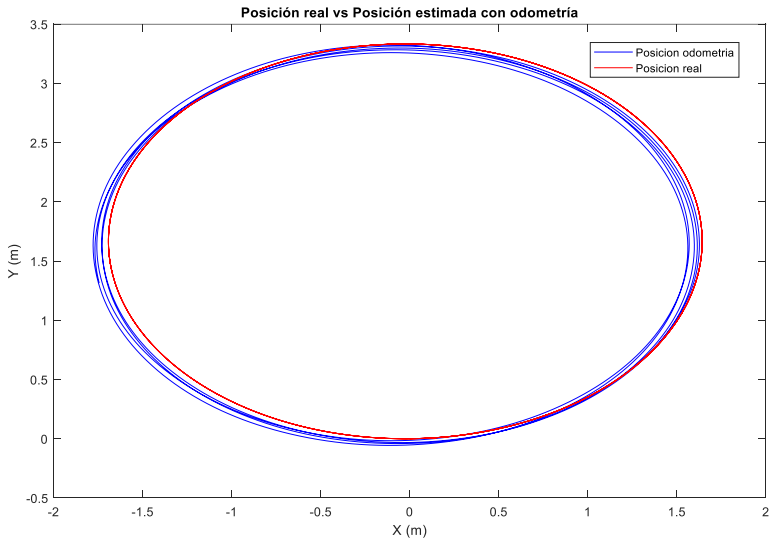

Figura 2. Representación de la posición real y de la posición estimada con la odometría en cada iteración.

5) A medida que se mueve el robot también se va calculando, para cada iteración, el avance y giro del robot con respecto a la iteración anterior, tanto los valores reales como los obtenidos según la odometría. Estos valores también se van almacenando para después poder conocer la incertidumbre cometida por la odometría en la estimación del avance y el giro del robot.

6) Se calcula el error absoluto cometido para cada variable de estado del vector de estado, es decir, el error cometido en *x*, el error cometido en *y* y el error cometido en θ. En la gráfica de la izquierda de la *Figura 3* se muestra la evolución de los errores en *x* e *y* mientras el robot describe la trayectoria elíptica. Por otro lado, en la gráfica de la derecha de la *Figura 3* se puede observar el error cometido en la estimación de la orientación θ.

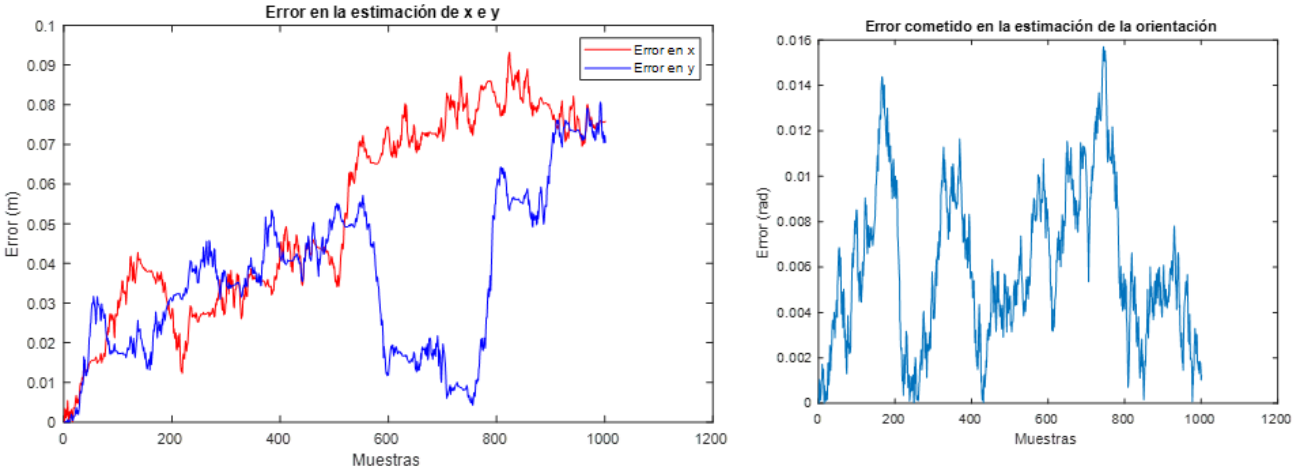

Figura 3. Error absoluto cometido en la estimación de [x y θ] para cada una de las 1000 muestras.

7) Se calcula el error cometido por la odometría al estimar el avance y el giro del robot durante su movimiento. En la gráfica de la izquierda de la *Figura* 5 se muestra el error absoluto del avance, mientras que en la de la izquierda se observar el error absoluto cometido en el giro.

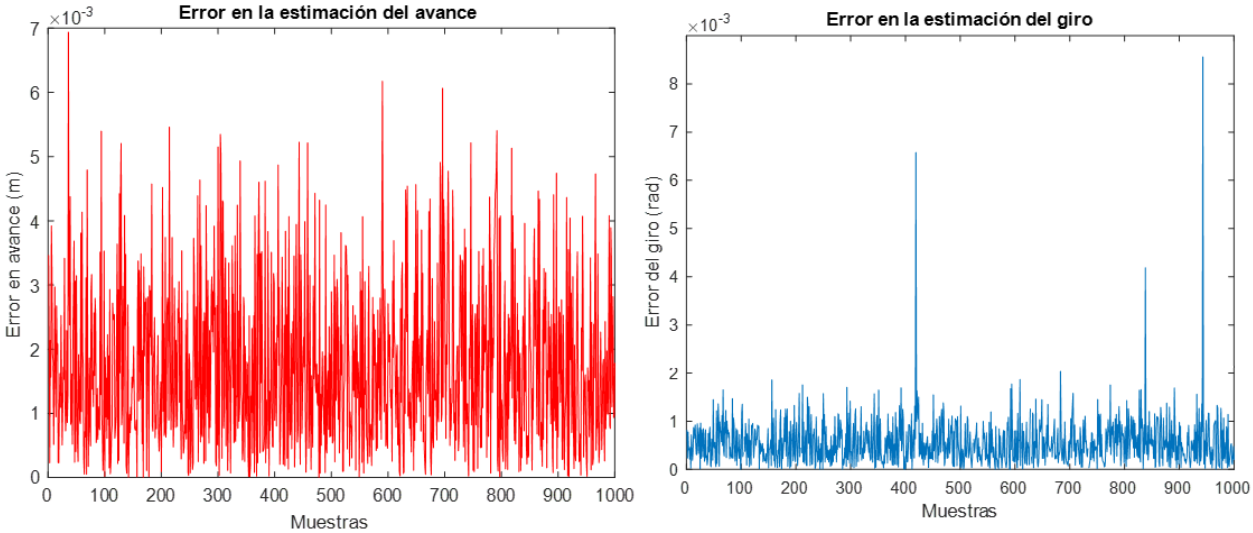

Figura 4. Errores cometidos en la estimación del avance y del giro según la odometría.

8) Finalmente se calculan las medias y las varianzas de cada uno de los errores calculados hasta ahora. Para ello, se ha programado un pequeño código en MATLAB que muestra por consola estos valores. En la *Figura 8* se muestra un ejemplo de los resultados que se obtuvieron con los parámetros de movimiento definidos en el punto 3 de este apartado.

```
--------------------------------------------------------------------------------
----------------INCERTIDUMBRE DEL MODELO CINEMÁTICO-----------------------

Valor medio de los errores absolutos cometidos con la odometría:
Error absoluto medio en coordenada x-------------> 0.05238 metros
Error absoluto medio en coordenada y-------------> 0.036449 metros
Error absoluto medio en la orientacion theta-----> 0.0059813 rad
Error absoluto medio en avance ------------------> 0.001731 metros
Error absoluto medio en el giro -----------------> 0.00056205 rad

Varianza correspondiente a cada variable de la odometría:
Varianza de la coordenada x----------> 0.00058603
Varianza de la coordenada y----------> 0.0003949
Varianza de la orientacion theta-----> 1.2028e-05
Varianza del avance------------------> 1.6375e-06
Varianza del giro --------------------> 2.7059e-07

--------------------------------------------------------------------------------
```

Figura 5. Medias y varianzas de los errores cometidos con la odometría.

Los valores de las varianzas de los errores absolutos cometidos en la estimación de las variables *x*, *y* y θ se utilizarán para la inicialización de la matriz P del filtro extendido de Kalman implementado en el apartado 5 de este documento. Esta matriz se corresponde con la matriz de varianzas y covarianzas de la estimación del estado (matriz de incertidumbres) e indica si la estimación es buena o no.

Por otro lado, los valores de las varianzas de los errores absolutos cometidos por la odometría en el avance y el giro del robot se emplearán para la inicialización de la matriz Q del filtro extendido de Kalman, ya que se corresponde con la varianza del ruido del proceso.

4 SISTEMA DE PERCEPCIÓN

En este apartado se especifican los sensores exteroceptivos que se utilizan para la localización y el control reactivo del robot. Para el caso de la localización se ha optado por utilizar un telémetro láser LMS100 y 10 balizas colocadas de forma concreta en las paredes del entorno. Por otro lado, para el caso del control reactivo se ha decidido utilizar 3 ultrasonidos, de esta manera se podrán detectar obstáculos que se encuentran cerca del robot y se podrán esquivar.

Para realizar la calibración y así poder estimar la incertidumbre de estos sensores se ha seguido el siguiente procedimiento:

1) Se coloca el robot en la posición [-2, 0] con una orientación de π rad. De esta manera, el robot se encuentra cerca de la mesa donde se atiende a los clientes del restaurante y mirando hacia ella. Esto permite calibrar los ultrasonidos.
2) Se colocan 2 balizas de tal manera que sean visibles por el telémetro láser del robot. En la *Figura 6* se puede observar la colocación del robot en el mapa del entorno y la localización de estas dos balizas.

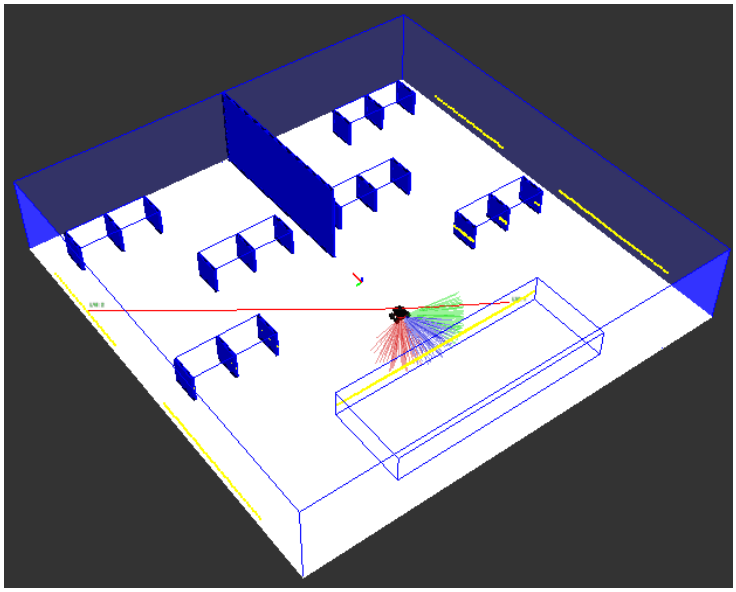

Figura 6. Posición del robot para la calibración de los sensores.

3) Se mide 3000 veces con cada uno de los tres ultrasonidos la distancia a la que se encuentra la mesa respecto del robot. Además, también se mide 3000 veces la distancia y el ángulo a la que se encuentra cada una de las dos balizas respecto del robot.

4) Se calcula la media y la varianza de las 3000 medidas realizadas por cada sensor. Los resultados obtenidos se muestran en la *Figura 7*. Como se puede observar en dicha figura, la varianza de los ultrasonidos es insignificante, del orden de 10^{-30}. En cuanto a los valores de las varianzas del ángulo y de la distancia del telémetro láser, estos valores se han utilizado para definir la matriz R de varianzas y covarianzas de la predicción de la medida del láser correspondiente al filtro extendido de Kalman.

```
----------------CALIBRACION DE LOS ULTRASONIDOS-------------------------

Valor medio de las distancias medidas por los ultrasonidos:
Distancia media del ultrasonidos frontal-----------> 1.8037 metros
Distancia media del ultrasonidos de la izquierda---> 1.8634 metros
Distancia media del ultrasonidos de la derecha-----> 1.8634 metros

Varianza correspondiente a las medidas de cada ultrasonidos:
Varianza del ultrasonidos frontal-----------> 5.9677e-30
Varianza del ultrasonidos de la izquierda---> 5.7014e-29
Varianza del ultrasonidos de la derecha-----> 5.7014e-29

----------------CALIBRACION DEL TELEMETRO LASER-------------------------

Valor medio de las distancias medidas por el laser a cada baliza:
Distancia media a la baliza LM1-----> 3.4991 metros
Distancia media a la baliza LM2-----> 9.9813 metros

Valor medio del ángulo al que se sitúan las balizas respecto el laser:
Angulo medio respecto a la baliza LM1----> 1.0302 rad
Ángulo medio respecto a la baliza LM2----> -2.2281 rad

Varianza correspondiente a las medidas de distancias hechas por el laser:
Varianza de la medida de distancia a la baliza LM1-----> 0.00028336
Varianza de la medida de distancia a la baliza LM2-----> 0.00027347

Varianza correspondiente a las medidas de ángulos hechas por el laser:
Varianza de la medida de ángulo a la baliza LM1-----> 0.00019857
Varianza de la medida de ángulo a la baliza LM2-----> 0.00018992
```

Figura 7. Ejemplos de valores obtenidos en la calibración de los ultrasonidos y el telémetro láser.

5 ESTIMACIÓN DEL ESTADO

Para poder implementar algoritmos de control y planificación de trayectorias es necesario estimar el estado del robot a medida que se va moviendo por el entorno. En este trabajo, para realizar esta estimación se emplea el filtro extendido de Kalman.

A continuación, se explica el procedimiento que se ha seguido para la implementación en MATLAB del algoritmo de localización que utiliza este filtro.

1) Inicialización de los parámetros del filtro.

- Se coloca el robot con una pose inicial determinada y se resetea la odometría para esa pose.
- Se inicializa la estimación del estado $\hat{x}_K$ con la función *apoloGetOdometry().*
- Se inicializa la matriz P_K de varianzas y covarianzas de la estimación del estado. Para ello, se utilizan los valores de las varianzas de los errores de estimación de *x*, de *y* y de θ, obtenidos en la calibración del sistema de locomoción (apartado 3 de este documento).

$$P_K = \begin{pmatrix} 5.8603 \cdot 10^{-4} & 0 & 0 \\ 0 & 3.949 \cdot 10^{-4} & 0 \\ 0 & 0 & 1.2028 \cdot 10^{-5} \end{pmatrix}$$

- Se inicializa la matriz *R* de varianzas y covarianzas de la predicción de la medida a partir de las varianzas obtenidas en la calibración del sensor láser (apartado 4 de este documento). Cabe destacar que para el modelo de observación se necesita detectar dos balizas, utilizando para la predicción de la medida los ángulos y una de las distancias que estas proporcionan. De esta manera, conociendo la posición de cada una de las balizas detectadas se pueden calcular las medidas que el telémetro laser debería indicar.

$$R_K = \begin{pmatrix} 1.9857 \cdot 10^{-4} & 0 & 0 \\ 0 & 1.9857 \cdot 10^{-4} & 0 \\ 0 & 0 & 2.8336 \cdot 10^{-4} \end{pmatrix}$$

- Se inicializa la matriz *Qk* correspondiente a la varianza en la medida. Para ello, se han utilizado los valores de las varianzas de los errores de avance y giro obtenidos en la calibración de la odometría (apartado 2 de este documento).

$$Q_K = \begin{pmatrix} 1.635 \cdot 10^{-6} & 0 \\ 0 & 2.7059 \cdot 10^{-7} \end{pmatrix}$$

- Se almacena el valor de las 10 balizas que se han colocado en el mapa del entorno. Estas balizas se han situado en zonas concretas del mapa de manera que el robot pueda ver al menos 2 de ellas la mayor parte del tiempo que se encuentra en movimiento.

2) Movimiento del robot y detección de balizas.

Después de la inicialización de los parámetros del filtro, se ejecuta el algoritmo de localización en bucle. Dentro del bucle, lo primero que se hace es mover el robot con la función *apoloMoveMRobot()* y se comprueba el número de balizas que el láser detecta con la función *apoloGetLaserLandMarks()*. Si se detectan al menos 2 balizas, entonces se ejecuta el código correspondiente al filtro extendido de Kalman. En caso contrario, el robot se sigue moviendo hasta detectarlas y únicamente predice la posición por medio de la odometría, es decir, no utiliza la corrección del filtro.

3) Predicción del estado

Para la predicción del estado ($\widehat{pose}_K^-$) se utiliza el modelo cinemático del sistema de locomoción de tipo diferencial:

$$\widehat{pose}_K^- = \begin{pmatrix} \hat{x}_K^- \\ \hat{y}_K^- \\ \hat{\theta}_K^- \end{pmatrix} = \begin{pmatrix} \hat{x}_{K-1} \\ \hat{y}_{K-1} \\ \hat{\theta}_{K-1} \end{pmatrix} + \begin{pmatrix} avance \cdot \cos\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ avance \cdot \text{sen}\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ \hat{\theta}_{K-1} + giro \end{pmatrix}$$

Esto significa que la predicción del estado depende de la estimación anterior y del avance y giro que ha realizado el robot según lo que mide la odometría. Para calcular el avance y giro respecto la pose anterior se utiliza la función *apoloGetOdometry()*.

Por otra parte, para conocer la matriz P_K^- de varianzas y covarianzas de la predicción del estado se utiliza la siguiente ecuación:

$$P_K^- = \phi_{K-1} \cdot P_{K-1} \cdot \phi_{K-1}^T + G_{K-1} \cdot Q_{K-1} \cdot G_{K-1}^T$$

Donde ϕ_{K-1} y G_{K-1} se corresponden con matrices jacobianas de la matriz $\widehat{pose}_K^-$, debido a que se trata de un modelo no lineal y hay que linealizarlo. La matriz jacobiana ϕ_{K-1} se obtiene al derivar las ecuaciones del modelo cinemático con respecto a las variables de estado (*x*, *y*, θ), mientras que la matriz jacobiana G_{K-1} se halla derivando las ecuaciones del modelo cinemático con respecto a las variables *avance* y *giro*. El resultado obtenido de realizar estas derivadas es el que se muestra a continuación.

$$\phi_{K-1} = \begin{pmatrix} 1 & 0 & -avance \cdot \text{sen}\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ 0 & 1 & avance \cdot \cos\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ 0 & 0 & 1 \end{pmatrix}$$

$$G_{K-1} = \begin{pmatrix} \cos\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) & -0.5 \cdot avance \cdot \text{sen}\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ \text{sen}\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) & 0.5 \cdot avance \cdot \cos\left(\hat{\theta}_{K-1} + \frac{giro}{2}\right) \\ 0 & 1 \end{pmatrix}$$

4) Predicción de la medida del sensor láser.

Como se ha comentado anteriormente, para la localización del robot, el láser lo que hace es medir los dos ángulos a los que se sitúan dos balizas respecto del robot y la distancia de una de ellas. Para la predicción de la medida del telémetro láser en función de la predicción del estado, se ha utilizado el siguiente modelo matemático:

$$\hat{z}_K^- = \begin{pmatrix} arctg\left(\frac{y_{baliza-1} - \hat{y}_K^-}{x_{baliza-1} - \hat{x}_K^-}\right) - \hat{\theta}_K^- \\ arctg\left(\frac{y_{baliza-2} - \hat{y}_K^-}{x_{baliza-2} - \hat{x}_K^-}\right) - \hat{\theta}_K^- \\ \frac{(y_2 - y_c)}{sen\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right)} \end{pmatrix}$$

Como se puede observar, la predicción de la medida, además de depender de la predicción del estado, también depende de las coordenadas *x* e *y* de las balizas que se están detectando por el láser. Por este motivo, en el algoritmo de Matlab, en la inicialización de las posiciones de las balizas se ha utilizado una matriz de

10x2, donde cada columna representa la coordenada *x* e *y,* y cada fila la posición de una baliza, de tal manera que se han colocado en orden creciente del parámetro "id".

Gracias a esto, como la función *apoloGetLaserLandMarks()* también obtiene el "id" de las balizas detectadas, se puede conocer la posición de cada una de estas.

Para calcular la matriz de observación que relaciona el estado con la medida es necesario calcular el jacobiano de la matriz $\hat{z}_K^-$ con respecto a la predicción del estado, es decir, con respecto a las variables $\hat{x}_K^-$, $\hat{y}_K^-$ y $\hat{\theta}_K^-$. Si se realizan estas derivadas parciales se obtiene la siguiente matriz jacobiana de observación:

$$H_k = \begin{pmatrix} \frac{y_{baliza-1} - \hat{y}_K^-}{(x_{baliza-1} - \hat{x}_K^-)^2 + (y_{baliza-1} - \hat{y}_K^-)^2} & -\frac{x_{baliza-1} - \hat{x}_K^-}{(x_{baliza-1} - \hat{x}_K^-)^2 + (y_{baliza-1} - \hat{y}_K^-)^2} & -1 \\ \frac{y_{baliza-2} - \hat{y}_K^-}{(x_{baliza-2} - \hat{x}_K^-)^2 + (y_{baliza-2} - \hat{y}_K^-)^2} & \frac{x_{baliza-2} - \hat{x}_K^-}{(x_{baliza-2} - \hat{x}_K^-)^2 + (y_{baliza-2} - \hat{y}_K^-)^2} & -1 \\ \frac{-(y_2 - y_c) \cdot \cos\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right) \cdot \frac{1}{1 + \left(\frac{y_2 - y_c}{x_2 - x_c}\right)^2} \cdot \left(\frac{y_2 - y_c}{(x_2 - x_c)^2}\right)}{sen\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right)^2} & \frac{-sen\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right) - (y_2 - y_c) \cdot \cos\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right) \cdot \frac{1}{1 + \left(\frac{y_2 - y_c}{x_2 - x_c}\right)^2}}{sen\left(arctg\left(\frac{y_2 - y_c}{x_2 - x_c}\right)\right)^2} & 0 \end{pmatrix}$$

5) Observación de la medida del sensor láser y comparación.

En esta parte del algoritmo se almacenan los valores de los ángulos y una de las distancias a las que se sitúan las dos balizas respecto del robot, según el telémetro láser ($\bar{z}_K$), con el fin de comparar estos valores con los que se han predicho en el paso anterior con la matriz $\hat{z}_K^-$. Si tras esta comparación el error obtenido es superior a un umbral, entonces se descarta la medida y no se utiliza para la corrección de la estimación del estado. En caso contrario, se calcula la ganancia del estimador W_K y la matriz S_K de la siguiente manera:

$$S_K = H_K \cdot P_K^- \cdot H_K^T + R$$
$$W_K = P_K^- \cdot H_K^T \cdot S_K^{-1}$$

6) Corrección de la medida de la predicción del estado.

En caso de que el error obtenido tras la comparación realizada en el paso anterior esté dentro del umbral establecido, entonces se procede a corregir la estimación del estado y su matriz de varianzas y covarianzas mediante las siguientes ecuaciones:

$$\widehat{pose}_K = \widehat{pose}_K^- + W_K \cdot (\bar{z}_K - \hat{z}_K^-)$$
$$P_K = P_K^- - W_K \cdot S_K \cdot W_K^T$$

Para comprobar si el algoritmo funciona correctamente, se ha programado que el robot móvil describa una determinada trayectoria en el simulador Apolo. Además, se ha ido almacenando en cada iteración del movimiento la estimación del estado obtenida con el filtro extendido de Kalman y la pose real del robot obtenida con la función *apoloGetLocation()*.

De esta manera, se ha podido representar en una gráfica de MATLAB los diferentes puntos de la trayectoria que ha seguido según la estimación del filtro y según Apolo, para poder establecer una comparación visual. Por último, también se han representado las varianzas acumuladas de la estimación del estado.

Los resultados obtenidos se presentan las *Figuras 8* y *9*. Como se puede observar en esas gráficas, los resultados obtenidos de la estimación son bastante acertados.

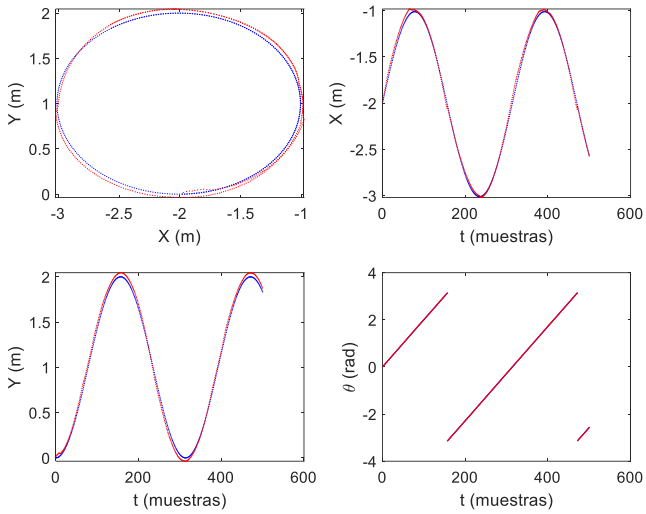

Figura 8. Comparaciones gráficas entre la estimación del estado y la pose real.

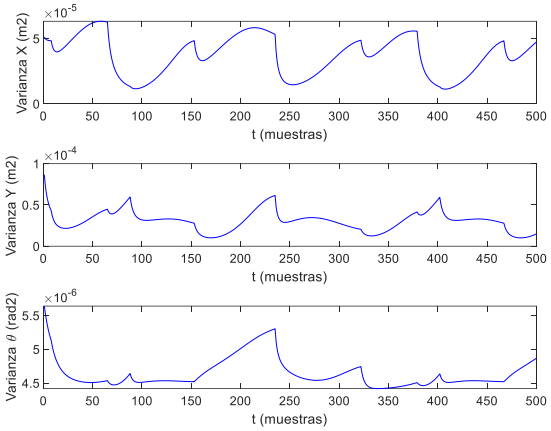

Figura 9. Varianza de x, y y theta durante las 500 iteraciones del bucle.

6 CONTROL

6.1 ALGORITMOS DE SEGUIMIENTO DE TRAYECTORIAS

Para llevar a cabo el movimiento del robot a lo largo de una trayectoria generada por un planificador, es necesario programar un controlador que vaya siguiendo los distintos puntos de la trayectoria. El sistema de control utilizado se denomina "*Pure Pursuit Algorithm*" y está implementado el Matlab, lo cual facilita su implementación en el trabajo.

Sin entrar en mucho detalle, el algoritmo mencionado consta de las siguientes etapas:

1. En primer lugar, se obtiene la pose del vehículo a través del algoritmo de localización que se ha programado previamente.
2. A continuación, se busca el punto en la trayectoria que se desea seguir, que se encuentra a una distancia determinada "*look-ahead distance*". La determinación de este punto se hace a partir de cálculos geométricos.

3. Posteriormente, se calcula la velocidad de giro necesaria para llegar al punto deseado, finalizando la iteración.

Dado que se debe especificar una velocidad de avance constante, el algoritmo de control trabaja únicamente con la velocidad de giro. De esta manera, existen dos parámetros que se pueden modificar para mejorar el comportamiento del controlador: la distancia "*look-ahead*" y la velocidad de avance.

6.2 IMPLEMENTACIÓN DEL ALGORITMO PURE PURSUIT

La programación de este algoritmo es sencilla ya que existe una clase en Matlab denominada "*controllerPurePursuit*" que contiene los métodos para la estimación de la velocidad angular que debe tener el robot.

Los parámetros que hay que definir para inicializar el controlador son: la trayectoria, la velocidad angular máxima, la velocidad de avance del robot y la distancia "*look-ahead*". Además, hay que especificar el error máximo permitido, ya que el algoritmo programado no llega al punto deseado, sino que se queda en una zona del entorno.

Se ha generado una trayectoria sinusoidal que debe seguir el robot y se ha comprobado el funcionamiento del algoritmo variando los parámetros de velocidad lineal y distancia *look-ahead*, obteniéndose las trayectorias mostradas en la *Figura 10*.

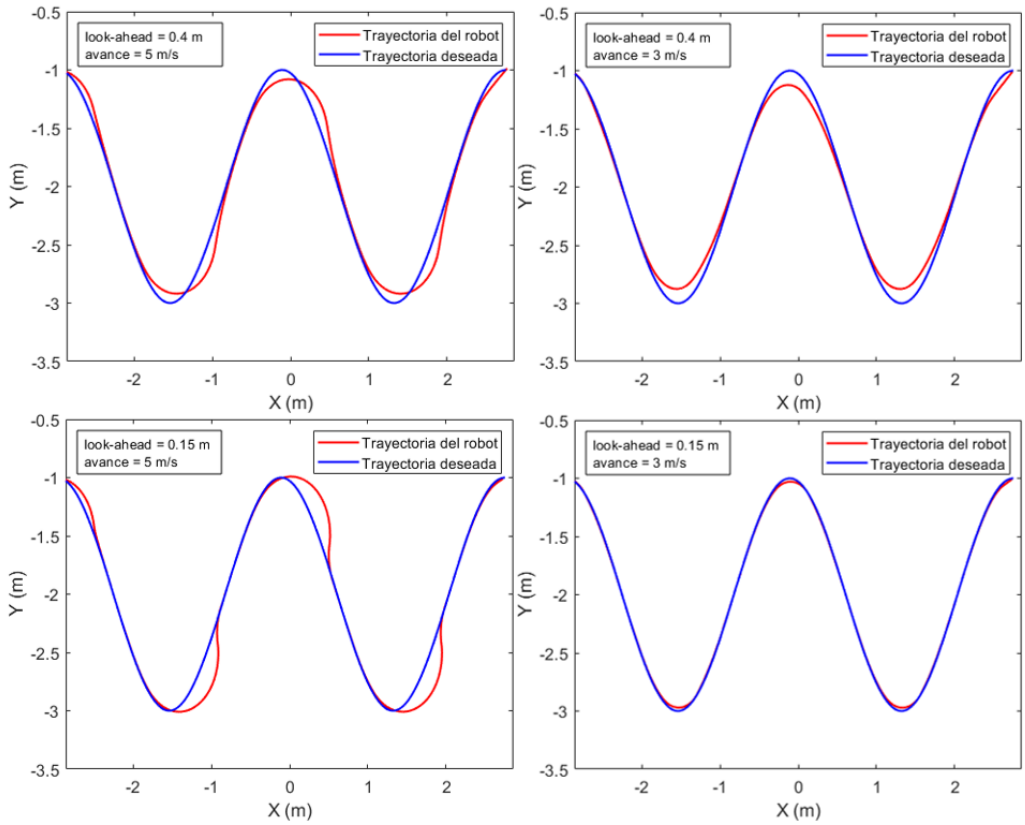

Figura 10. Trayectorias simuladas para distintos parámetros del controlador.

Como se puede observar, la elección de los valores del controlador es esencial para su correcto funcionamiento. Por otro lado, si se escoge una velocidad de avance elevada, el control se dificulta a la hora de realizar giros bruscos, por lo que es conveniente mantener una velocidad de avance limitada.

6.3 IMPLEMENTACIÓN DEL CONTROL REACTIVO

La función del control reactivo consiste en evitar colisiones con obstáculos que se presenten en el camino. Es un método de control que otorga cierta autonomía al robot sin un conocimiento a priori del entorno donde se encuentre. El principal problema de este método de control es que se pueden alcanzar mínimos locales de los cuales no se pueda salir fácilmente.

Por lo tanto, la arquitectura de control reactiva entra en ejecución cuando se detecta un obstáculo cercano. Estos obstáculos no se han tenido en cuenta en la planificación porque no se encuentra esa información en el mapa del entorno. Una causa son los obstáculos móviles, como el paso de personas.

Como es lógico, es necesario tener en cuenta estos obstáculos para mantener un control seguro del robot. Este tipo de control se encuentra, por tanto, en un nivel de autoridad superior que el control jerárquico y actuará únicamente cuando se detecten obstáculos que superen cierta distancia de seguridad.

Los sensores utilizados en este trabajo para que gobiernen el control reactivo son sensores de ultrasonidos, debido a que se encuentran disponibles en Apolo su implementación es muy sencilla. Además, en el proceso de la calibración se observa que la varianza de los ultrasonidos es del orden de los $10^{-30}m^2$ por lo que se puede asegurar (en este caso de uso) que su incertidumbre asociada es 0.

Se ha optado por usar tres sensores de ultrasonidos en la parte delantera del robot, de forma que uno gobierne los obstáculos que se puedan encontrar de frente, otro los obstáculos a la derecha y otro a la izquierda (ver *Figura 11*). Así se consigue una mayor precisión del entorno y por tanto un control más sencillo. Debido a que el uso de varios ultrasonidos puede producir interferencias entre ellos, se han separado una cierta distancia para solventar este problema.

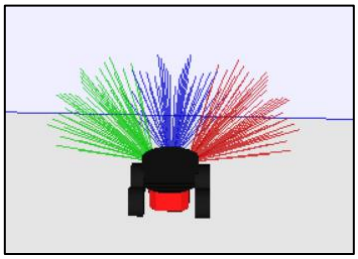

Figura 11. Uso de tres ultrasonidos para detección de obstáculos.

El código se ha realizado en MATLAB y contempla una condición de precaución, cuando se detecte algún obstáculo a menos de 0.8 metros, y se impone una velocidad de giro y avance determinadas. En caso de que esa distancia de seguridad se vea reducida a 0.3 metros o menos, se detiene el avance del robot imponiendo únicamente una velocidad de giro. Debido a la naturaleza de este control, para escoger los valores de velocidades impuestas, se ha llevado a cabo un proceso iterativo.

La principal característica de este código es que se imponen unas velocidades en función de las distancias medidas por los ultrasonidos. La velocidad de avance será proporcional con la distancia frontal, de modo que a mayor distancia a los obstáculos mayor rapidez en el avance. El razonamiento para la velocidad de giro es el opuesto, inversamente proporcional con las distancias medidas a los lados. Así se consigue evitar el obstáculo más rápido cuanto más pequeña sea la distancia medida.

Para comprobar el código elaborado con los parámetros finalmente escogidos, se ha ejecutado varias veces sobre el entorno diseñado añadiendo diferentes obstáculos estáticos y móviles (otro Pioneer3AT también con control reactivo). Finalmente, los resultados han sido satisfactorios consiguiendo un control autónomo sin que su movimiento se encuentre impedido por obstáculos en su trayectoria.

7 PLANIFICACIÓN DE TRAYECTORIAS

Para poder llegar desde un punto inicial hasta un punto final sin chocar con ningún obstáculo en el camino, es necesario generar una serie de puntos en el entorno que determinen la trayectoria que debe seguir el robot para lograr su objetivo.

Existen múltiples algoritmos para obtener los puntos deseados, entre los cuales destacan los planificadores discretos, los planificadores basados en combinatoria y los planificadores basados en muestreo. Entre todas las opciones, se ha escogido un algoritmo RRT que construye árboles de exploración para encontrar la trayectoria deseada.

El método RRT está basado en muestreo y está incluido en Matlab, facilitando su implementación para el trabajo. Además, presenta la gran ventaja de trabajar en el espacio de acciones en vez de en el espacio de estados, manteniendo las restricciones del robot no holonómico con el que se trabaja. Esto último asegura que los puntos generados por la trayectoria se pueden alcanzar.

Sin embargo, este algoritmo presenta el problema de que no se tiene la orientación inicial del robot, pudiendo dar lugar a un inicio de la trayectoria no realizable desde la pose inicial. Este problema se puede solucionar desde la etapa de control, pero sería deseable que el planificador tuviera en cuenta esta situación y pudiera planificar una trayectoria posible para cualquier orientación inicial.

7.1 IMPLEMENTACIÓN DEL ALGORITMO RRT

Para llevar a cabo la planificación, en primer lugar se debe importar el mapa del entorno de trabajo, que se ha obtenido a partir de la planta del restaurante. A continuación, dado que el robot tiene una anchura determinada y el planificador trabaja con puntos discretos, es necesario ensanchar los obstáculos para evitar que el robot colisione con ellos. Este ensanchamiento toma como valor la mitad del ancho del robot. Después de esto, se genera un mapa de ocupación de celdillas y se escala para coincidir con las dimensiones del entorno.

Una vez establecido el mapa, se genera un espacio de estados [x, y, θ] utilizando la función de Matlab "*stateSpaceSE2*". Dicho espacio de estados contiene la física involucrada en el movimiento del robot basándose en distancias euclídeas e interpolaciones lineales para calcular traslaciones y rotaciones.

Por otro lado, hay que establecer un elemento denominado "*validador*" que se encarga de detectar las posibles colisiones que se produzcan durante la generación de la trayectoria. Este objeto comprueba la validez de cada estado generado por el planificador con una resolución determinada por el usuario.

Tras esto, se crea el planificador a partir del espacio de estados y del validador. Además, se debe establecer la distancia máxima que tendrán las aristas de los árboles generados y el error permitido en el punto final planificado respecto al punto final deseado. Finalmente, se ejecuta el planificador y se obtiene la trayectoria de puntos que llevan al robot móvil desde la posición inicial hasta la posición final.

7.2 RESULTADOS DE LA PLANIFICACIÓN

Tras un proceso iterativo de ajuste, se ha obtenido los parámetros mostrados en la *Tabla 1.*

Dimensiones del mapa de ocupación	[0, 0] – [16, 16] m
Ancho del robot	0.497 m
Distancia de detección de colisión	0.001 m
Máximo tamaño de arista del árbol de búsqueda	0.2 m
Error máximo permitido (Distancia entre el punto final planificado y el deseado)	0.02 m

Tabla 1. Parámetros escogidos para el planificador.

Pese a que el algoritmo necesita un menor tiempo de ejecución si se escogen valores mayores de tamaño máximo de arista del árbol, se ha decidido elegir un valor de 0.2 m para reducir la posibilidad de que el robot quede atrapado debido a su naturaleza no-holonómica.

Para comprobar el funcionamiento del planificador, se han generado varias trayectorias a partir de distintos puntos de inicio y de destino. Sin embargo, como se ha explicado en el apartado de control, el sistema de referencia del mapa de ocupación y el del entorno en Apolo no coinciden, por lo que hay que llevar a cabo las transformaciones de coordenadas necesarias con las funciones "*apolo2map*" y "*map2apolo*".

Las pruebas realizadas para los parámetros escogidos se muestran en la *Tabla 2*, en la cual se han representado todas las coordenadas en el sistema de referencia del mapa para una mejor visualización.

Nº Prueba	Pose Inicial [m, m, rad]	Punto Final [m, m]	Tiempo de ejecución (s)
1	$[8, 8, \pi/2]$	[5 10]	0.04 – 0.07
2	$[10.7, 0.5, 0]$	[5 10]	0.05 – 0.12
3	$[10.7, 15.5, 0]$	$[10.7, 0.5]$	0.13 – 0.19

Tabla 2. Resultados de la planificación de trayectoria.

La representación gráfica de las trayectorias planificadas así como de los árboles de búsqueda resultantes del algoritmo RRT se pueden observar en la *Figura 12*.

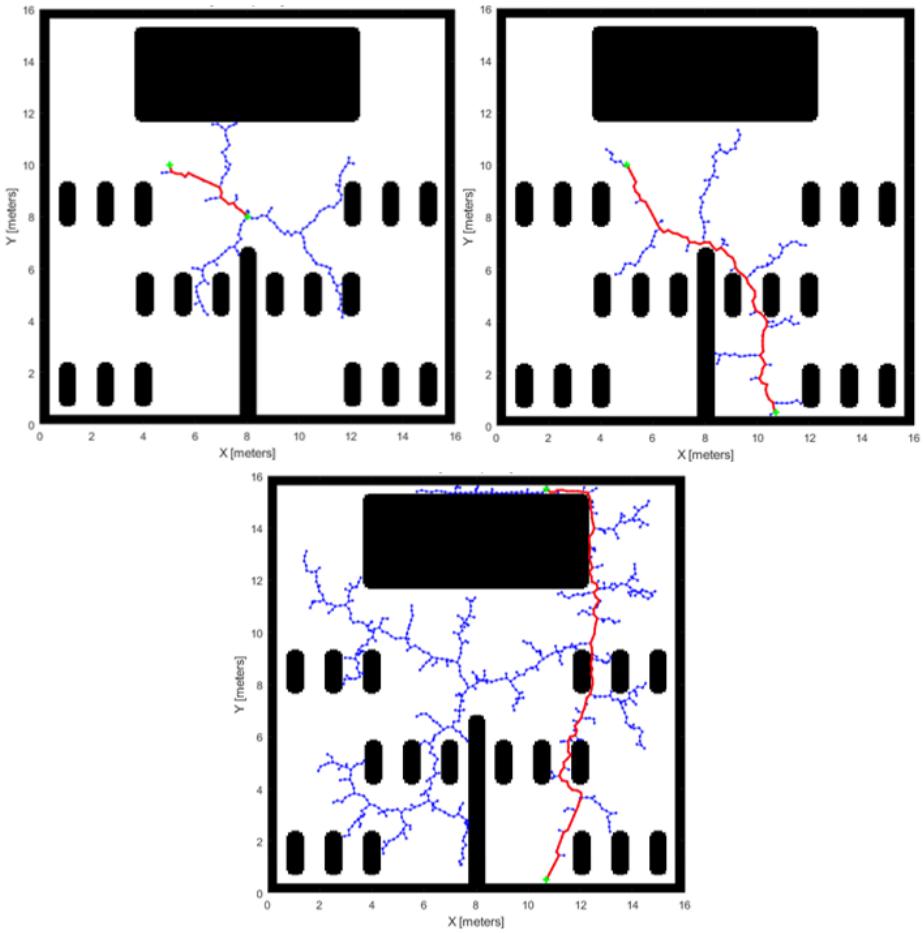

Figura 12. Trayectorias generadas para las pruebas 1, 2 y 3 de la Tabla 2.

8 DEMOSTRADOR FINAL

Una vez que se han diseñado los tres bloques para el guiado y navegación del robot móvil, se debe hacer la implementación conjunta del sistema completo y corregir los posibles errores que puedan surgir durante el proceso. El funcionamiento completo del sistema se muestra en la *Figura 13.*

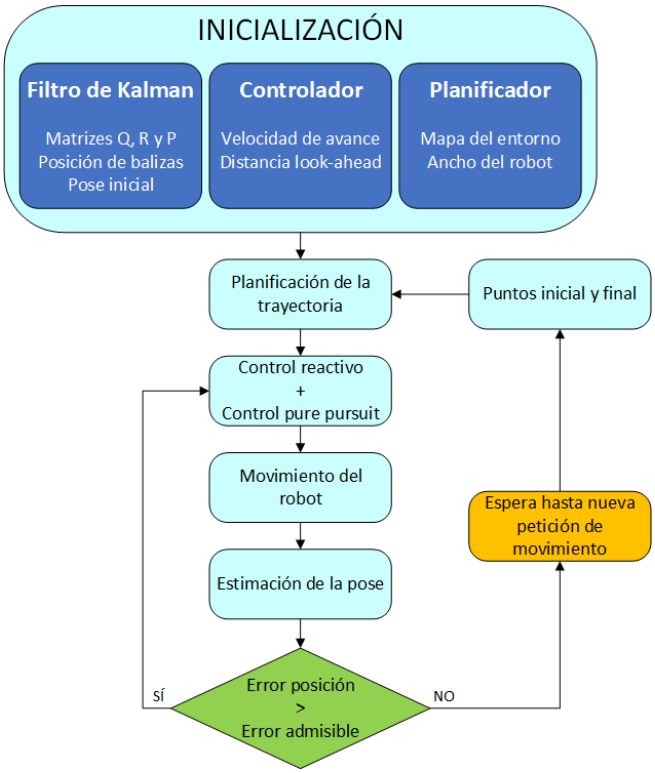

Figura 13. Diagrama de flujo del guiado y navegación del robot móvil.

El primer paso es la inicialización de los distintos parámetros de cada bloque. En el filtro de Kalman se deben establecer los valores iniciales de las matrices de varianzas y covarianzas, la pose inicial del robot y la posición de las balizas. En cuanto al controlador, se inicializan los valores de la velocidad de avance y de la distancia *look-ahead*. Para la inicialización del planificador hay que cargar el mapa del entorno y el ancho del robot para su ensanchamiento.

Una vez se han inicializado los valores de cada bloque, se establece un punto inicial y un punto final para introducir en el planificador, que genera una trayectoria de puntos utilizando el algoritmo RRT.

A continuación, se entra en un bucle de seguimiento de trayectoria en el cual se genera una consigna de movimiento en función del control (prevaleciendo siempre el control reactivo sobre el de seguimiento de trayectoria). Para comprobar el error de seguimiento, se debe estimar en cada iteración la pose del robot utilizando el filtro de Kalman. En el caso de que no se observen dos balizas simultáneamente se tomará el valor de la odometría directamente sin el filtro.

Cuando la distancia del robot a la posición final deseada se reduce hasta un umbral admisible, se considera que el robot ha llegado al punto de destino y finaliza la ejecución del programa. El bloque de espera representa el funcionamiento que tendría una implementación real en el robot móvil.

Para comprobar el correcto funcionamiento del sistema completo, se han realizado varias simulaciones con distintos puntos de inicio y de fin, contrastando la trayectoria planificada con la trayectoria estimada del filtro de Kalman y con la trayectoria real que ha recorrido el robot.

Los puntos escogidos para las simulaciones se encuentran recogidos en la *Tabla 4.*

	Pose Inicial [m, m, rad]	Punto Final [m, m]
Simulación Nº1	$[6.5, -2.7, \pi]$	$[6.5, 2.7]$
Simulación Nº2	$[6.5, 2.7, \pi]$	$[-6.5, -6.5]$
Simulación Nº3	$[-6.5, 6.5, 0]$	$[6.5, -2.7]$
Simulación Nº4	$[-6.5, -6.5, 0]$	$[-6.5, 6.5]$

Tabla 3. Datos de las simulaciones para el demostrador.

Al simular el sistema completo para los cuatro pares de puntos, se obtienen las trayectorias mostradas en la *Figura 14.* Para mejorar la visualización de la trayectoria, se ha invertido el eje X al realizar las gráficas.

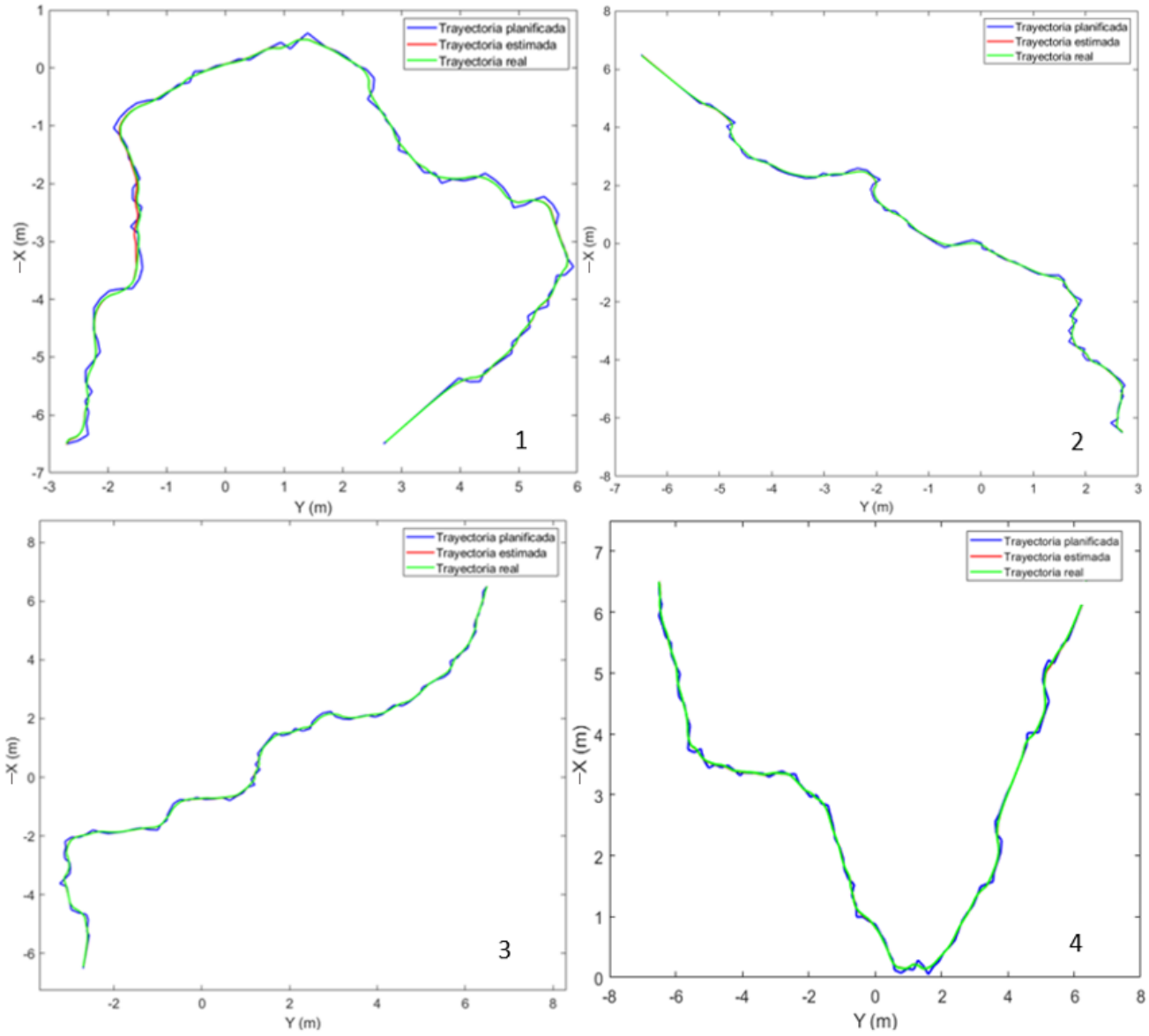

Figura 14. Trayectorias del demostrador para las cuatro simulaciones.

Debido al buen comportamiento del filtro de Kalman, la trayectoria real y la estimada se superponen en casi todo momento, salvo en un intervalo en la primera simulación en la que no se llegan a detectar dos balizas, generándose un error mayor debido a la incertidumbre asociada a la odometría del robot.

Además, se han representado las trayectorias planificadas sobre el mapa del entorno para una mejor visualización de cada simulación en la *Figura 15.* Hay que tener en cuenta la transformación de coordenadas que existen entre el Apolo y el mapa generado en Matlab, de tal forma que la *Figura 15* busca aportar información visual y no numérica.

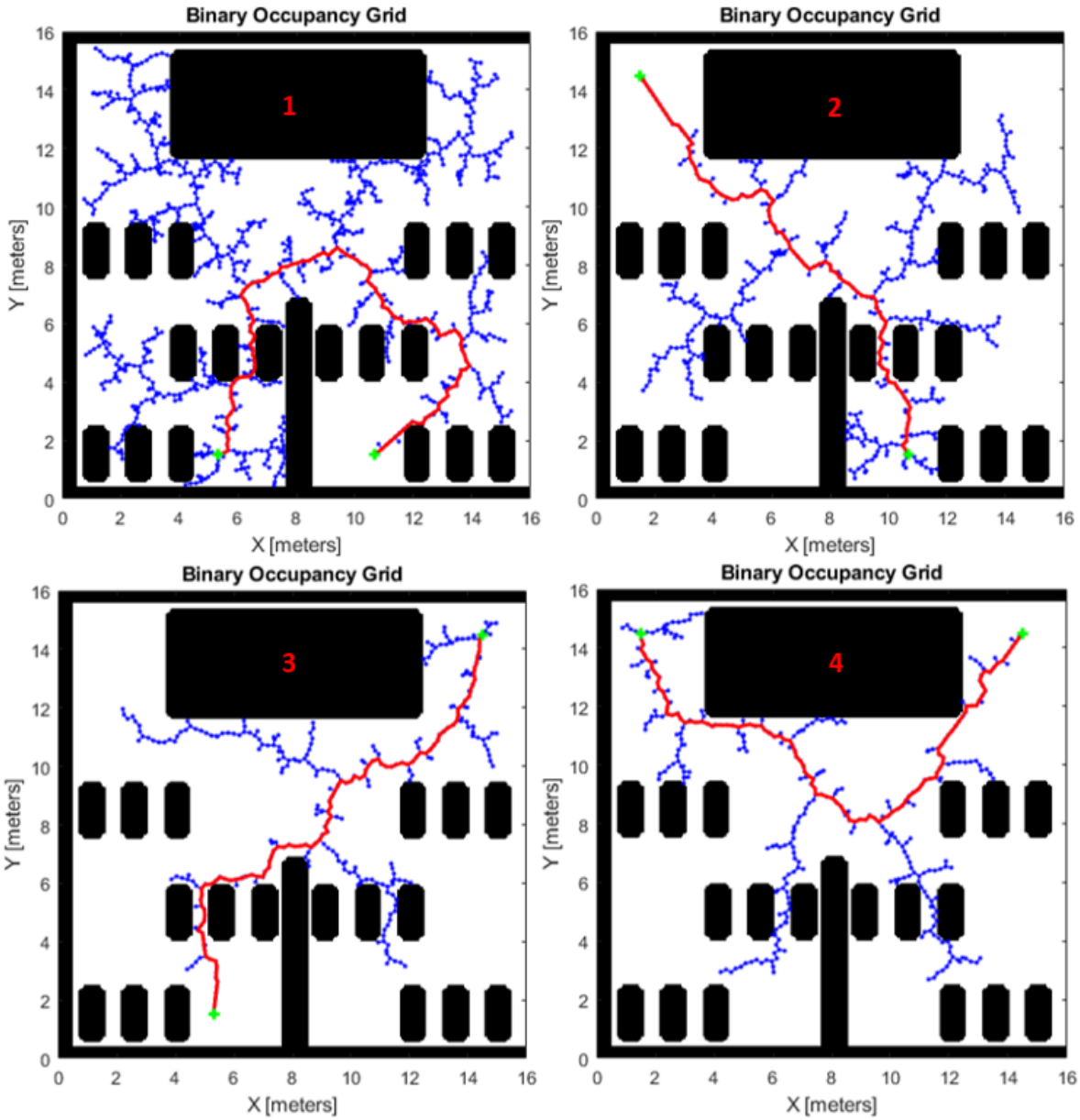

Figura 15. Trayectorias planificadas para cada simulación.

9 CONCLUSIONES Y OBSERVACIONES

A continuación, se exponen las principales conclusiones y observaciones extraídas de este trabajo.

La calibración del telémetro láser ha supuesto una parte bastante importante de este trabajo puesto que ha permitido ajustar los parámetros de la matriz R de varianzas y covarianzas de la estimación de la medida. Por otro lado, mediante la calibración de la odometría se ha podido inicializar la matriz P de varianzas y covarianzas de la estimación del estado, así como la matriz Q que representa la varianza del ruido del proceso. A partir de estos valores ya se han conseguido resultados muy satisfactorios en la implementación del filtro extendido de Kalman en el algoritmo de localización.

Con el filtro extendido de Kalman se ha conseguido ir corrigiendo el error cometido por la odometría en la estimación del estado del robot y así evitar que esta vaya acumulando cada vez un error mayor. En un principio, cuando se desarrolló el filtro extendido de Kalman, se optó por utilizar el ángulo de tres balizas para la estimación del estado. Sin embargo, en varias ocasiones no se llegaban a detectar tres balizas y no se podía corregir la medida de la odometría. Esto provocaba ciertos errores en los resultados obtenidos al comparar la pose estimada con la real. Para solucionar este problema, se decidió utilizar un modelo de observación que empleara únicamente dos balizas en vez de tres. De esta manera, se ha conseguido corregir la estimación del estado en más iteraciones que cuando se necesitaban tres, obteniendo menos error en la comparación entre la pose real y estimada.

En cuanto al algoritmo de control, se ha implementado un controlador básico y otro reactivo. El controlador básico ha permitido que el robot fuese capaz de seguir una trayectoria predefinida por un planificador, desde un punto inicial a otro final. Por otro lado, el controlador reactivo, a pesar de ser muy simple es bastante eficaz y proporciona de cierta autonomía al robot. Tanto es así, que se ha comprobado mediante diferentes experimentos. Por ejemplo, se han añadido varios robots Pioneer3AT y varios objetos estáticos en el entorno y se ha conseguido que el robot bordeara cualquier obstáculo que se encontraba mientras recorría la trayectoria indicada por el planificador, pudiendo llegar al punto final fijado como destino.

La planificación ha resultado sencilla de implementar gracias a los algoritmos incluidos en Matlab, pero conviene conocer cómo funcionan para saber cuál de ellos es el adecuado para la aplicación desarrollada. Por ejemplo, se ha podido observar la existencia de una componente aleatoria en el RRT ya que las trayectorias generadas entre dos puntos variaban en cada ejecución del programa. Esto significa que el algoritmo escogido no escoge el camino más corto, sino el primero que encuentra. Es un algoritmo cuyo criterio de busqueda de la trayectoria es la factibilidad sobre la optimilidad.