UNIVERSIDAD POLITÉCNICA DE MADRID
Máster en Automática y Robótica
Doble Máster en Ingeniería Industrial y Automática y Robótica
Guiado y Navegación de Robots
Grupo 21

ROBOT AYUDANTE PARA LA PODA DE SETOS

German Moreno Escudero (M20251), Jorge Deiro Ferre (M19056), Sergio Martínez Hamdoun (M19155)

Resumen
En este trabajo se han reforzado los temas principales que aparecen en la navegación de un robot en presencia de incertidumbre. Entre ellos se encuentran los sistemas de locomoción, elección de sensores y algoritmos de estimación del estado, de control de movimientos y planificación de trayectorias. Es por ello que se presenta una aplicación real donde se usarán estos sistemas en conjunto.

Índice

1. APLICACIÓN ESCOGIDA

1.1 Robot ayudante para la poda de setos

En la elaboración del trabajo, en primer lugar, comenzamos escogiendo la aplicación sobre la que queríamos hacer trabajar a nuestro robot. Para esta aplicación se elaborará el mapa del entorno sobre el entorno de simulación *Apolo* desarrollado por alumnos y profesores de la ETSIDI (UPM). En él podremos tener una visualización gráfica del robot y de sus sensores y podremos realizar la simulación del modelo cinemático del robot y de las medidas de los sensores. Para ello también se ha hecho uso del software *Matlab* que permite la implementación de los algoritmos de localización, control y planificación.

La aplicación elegida por lo tanto ha sido de un robot jardinero. Para ello se ha escogido una zona del Parque del Retiro de Madrid y se ha implementado en el entorno de *Apolo*. La idea de trabajo del robot es que sea capaz por medio de su odometría y de las balizas colocadas a lo largo del parque de irse moviendo adecuadamente para cortar los arbustos que vaya dejando atrás.

Figura 1. Imagen aérea real de la zona modelada del parque del Retiro

Se puede observar en la 2 el entorno simulado como no solo va a tener como obstáculos los arbustos sino que en el parque también encontramos otro tipo de objetos que tendrá que ser capaz de evitar como son farolas, papeleras, bancos, casetas e incluso fuentes.

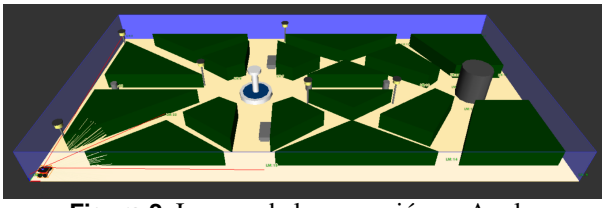

Figura 2. Imagen de la recreación en Apolo

Esta aplicación fue elegida por varios motivos. El primero de todos es el gran componente social que pueden llegar a tomar los robots en la vida de las personas en un futuro no muy lejano. Esta aplicación en principio puede no parecer tan común como el robot aspiradora que la gente pueda tener en casa o el robot móvil que nos podamos encontrar en un almacén y esa es la razón principal por la que decidimos llevarlo a esta aplicación, se trata de algo novedoso con posibilidad de volverse real en un tiempo cercano.

La segunda razón por la que decidimos tomar esta aplicación es por las dificultades que se puede encontrar el robot a lo largo de su acción. En esta, no solo tenemos muchos obstáculos que pueden hacer perder de vista las balizas por parte del robot, como son los arbustos, sino que también el control reactivo por el que el robot deberá esquivar los obstáculos de a pie, como son papeleras o bancos, hacen de esta aplicación una simulación perfecta para poder familiarizarnos con todos los temas que aparecen en la programación de la navegación de un robot en presencia de incertidumbre.

2. CALIBRACIÓN

En esta sección se va a tratar de los experimentos realizados para calibración y estimación de las matrices Q y R propias de nuestro vehículo y entorno, que serán vitales para el correcto funcionamiento de nuestro sistema de localización y estimación de la posición.

2.1 Sistema de locomoción

Para comenzar la calibración de nuestro robot, lo primero y más importante es identificar las características tanto de este como de los sensores que le vamos a introducir para que el funcionamiento del robot sea lo mejor y más preciso posible. El primer punto por lo tanto que hay que tratar es el sistema de locomoción de este, de que manera, nuestro robot *Marvin* se mueve por el parque.

Marvin se trata de un robot no holonómico ya que tiene menos grados de libertad controlables que el número total de grados de libertad posibles. Se puede mover por el plano con tres grados de libertad (x, y, theta) pero en cambio sólo podremos controlar la velocidad de avance y la velocidad de giro.

Para la calibración del comportamiento de vehículo con el entorno cuando trata de desplazarse, hay que tratar de ver si el sistema de odometría implementado está arrojándonos valores adecuados de desplazamientos al introducirle nosotros una acción.

Para este propósito, se realizó un campo de experimentación, donde se sometía al robot a una acción tanto de velocidad lineal como de velocidad angular. A partir de la distancia recorrida, y siendo conocido el sistema de locomoción del vehículo, se puede estimar cuales han sido las velocidades reales que se han introducido en el vehículo para alcanzar esta posición. En las ecuaciones 1 y 2.

$$v_{estimado,k} = \frac{X_{real,k} - X_{real,k-1}}{h * cos(\frac{\theta_{real,k}}{2} + \frac{\theta_{real,k-1}}{2})} \tag{1}$$

$$\omega_{estimado,k} = \frac{\theta_{real,k} - \theta_{real,k-1}}{h} \tag{2}$$

Tomando estos valores obtenidos y haciendo la diferencia con los valores que se han introducido en el sistema real, obtenemos el error que está cometiendo nuestra predicción respecto a la acciones que estamos dando. Esto es debido a que los motores pueden no estar transmitiendo al vehículo toda la potencia de desplazamiento que le indicamos por temas tales como la tracción de las ruedas con el terreno.

Tras la realización de estos experimentos, se perseguía la estimación de la matriz Q. Para ello se realizaron varias pruebas a velocidad lineal constante y velocidad angular constante. Los errores obtenidos, eran procesados y se calculaba la desviación típica de los mismos para cada una de las acciones. De esta manera se obtienen valores extremadamente pequeños de desviación típica del error. Estos valores son bastante pequeños, pero aún así se implementarán en la resolución, quedando la matriz Q como en la expresión 3.

$$Q = \begin{bmatrix} 1 * 10^{-15} & 0 \\ 0 & 1 * 10^{-16} \end{bmatrix} \tag{3}$$

El motivo por el que nuestro robot tiene un error de proceso nulo es que el simulador *Apolo* no presenta error en el movimiento del robot. Nuestro modelo de odometría utiliza los comandos de velocidad lineal y angular que se le piden al robot y calcula cuánto se moverá a partir de esos valores. Como *Apolo* no modela el error en el posicionamiento, los cálculos de odometría predicen perfectamente la ubicación en la siguiente iteración y el error que vemos es cero.

A continuación, y de la misma manera, se va a proceder a hacer la estimación de la matriz $P(0|0)$. Para ello se ha partido de los mismos experimentos realizados anteriormente, pero en vez de estimar las acciones U_k se ha estimado la posición

del vehículo real y la calculada por el modelo cinemático. A partir de este, se ha realizado la diferencia, y de igual manera se ha calculado la desviación típica del error. De esta manera, queda la matriz $P(0|0)$ como aparece en la expresión 4.

$$P_{0,0} = \begin{bmatrix} 0,013 & 0 & 0 \\ 0 & 0,010 & 0 \\ 0 & 0 & 0,010 \end{bmatrix} \quad (4)$$

Además, en la realización de estos experimentos, se notaron ciertos valores de saturación en los motores de *Marvin*. Para ello de la misma manera, se comprobó con diferentes acciones, cuales eran los que realmente se estaban dando. Es por ello que se estimaron valores de saturación de $v_{saturacion} = 2m/s$ y $w_{saturacion} = 1,0rad/s$. Estos valores se implementarán también en la estimación de la posición para evitar estimaciones incorrectas de la posición.

2.2 Sensores

Marvin, se va a mover libremente por el parque, con una trayectoria definida por el planificador que se describirá más adelante pero este va a tratarse de un robot móvil. Para poder localizarse, y sobre todo para poder esquivar los obstáculos que se vaya encontrando por el camino se ha decidido colocarle dos tipos de sensor. En primer lugar tenemos el sensor láser, la función principal de este será localizar las balizas que se han colocado en las diferentes esquinas del parque, su punto concreto ha sido previamente definido, de esta manera midiendo la distancia a esas balizas será capaz de situarse en el camino.

En segundo lugar, tenemos los sensores ultrasonidos, estos van a encargarse de lo que se encuentra cercano al robot, localizando los objetos que puedan obstruir el paso de este y en función de lo programado en el controlador, actuar. Se ha decidido colocarle 5 sensores ultrasonidos en la parte frontal, esto se debe a que tras una primera aproximación vimos como había zonas, como por ejemplo las esquinas, que los dos sensores que estaban colocados no eran capaces de detectarlas correctamente.

En primer lugar, se ha calibrado el sensor láser, el cuál su calibración era totalmente indispensable para el cálculo de la matriz *R* la cuál será necesaria para una correcta localización de *Marvin* en el mapa. Esta se implementará junto al *Filtro de Kalman* que posteriormente se explicará en más profundidad.

Para el cálculo de la dispersión del sensor láser en ángulo y en distancia, se ha dispuesto en un terreno de prueba donde se han realizado numerosas tomas de de estos valores. Conocido la posición real del vehículo, del sensor láser y de las balizas, se sabe exactamente estos valores, por lo que ha sido solo necesario realizar su resta para el calculo del error. Tras esto se ha calculado la desviación típica de este error, para finalmente montar la matriz *R*. Destacar que como comentaremos después, no se han tomado directamente los valores de ángulo, si no que se ha dividido en su seno y coseno, para conservar la continuidad en todo el rango y evitar saltos.

De esta manera, la matriz *R* queda de la manera indicada en la expresión 5.

$$R = \begin{bmatrix} 0,0136 & 0 & 0 \\ 0 & 0,0090 & 0 \\ 0 & 0 & 0,0100 \end{bmatrix} \quad (5)$$

El primer valor corresponde con la desviación típica del error de la distancia, el segundo con la del seno del ángulo y la tercera con la del coseno del ángulo.

3. LOCALIZACIÓN

En la localización, como se ha ido adelantando en puntos anteriores, se va a hacer de manera que se tenga en cuenta tanto la odometría como la percepción adquirida por nuestros sensores exteroceptivos. Para este propósito de ha implementado un *Filtro de Kalman* Extendido. Esto nos permite la introducción de sistemas no lineales como modelos de estado.

Este *Filtro de Kalman*, se divide en tres partes diferenciadas, como son, la obtención del modelo cinemático de nuestro vehículo, que en este caso *Marvin* se trata de un vehículo con cinemática diferencial, la obtención del modelo de observación, el cuál a partir de la localización de unas balizas con posición conocida, el vehículo es capaz de tomar información del entorno para una posible corrección de los valores dados por la odometría, y por último, una obtención de las matrices Jacobianas que permitirán realizar una estimación de los valores que se han de tomar para la fusión sensorial del mismo.

Además, como se ha venido comentado en anteriores apartados, se implementarán la estimación de las matrices R y Q, así como los valores de saturación de los motores estimados.

A continuación se va a desarrollar como se han implementado cada una de las partes de este sistema de localización.

3.1 Modelo cinemático

En primer lugar se ha debido realizar una estimación del modelo cinemático de *Marvin*. Este cuenta con sistema de locomoción diferencial, en el cuál, el movimiento conjunto de las ruedas derecha e izquierda, le permite la realización de diferentes desplazamientos lineales y angulares. En la Figura 3 se puede ver una representación de esto.

Con esta información, se pueden obtener las ecuaciones que modelan el movimiento de Marvin. Estas ecuaciones son las ecuaciones 6, 7 y 8.

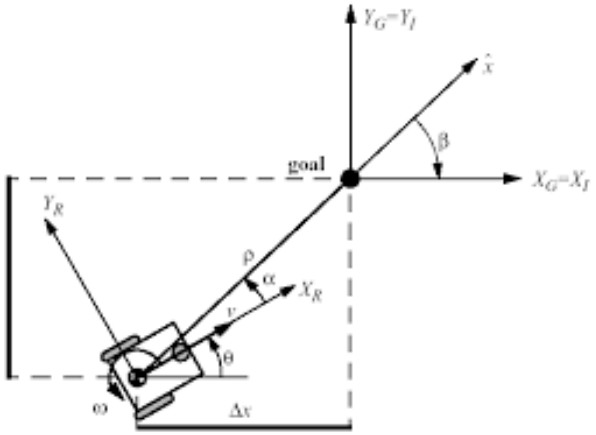

Figura 3. Representación del sistema de locomoción de Marvin

$$x_k = x_{k-1} + v * h * cos(\theta_{k-1} + \frac{wh}{2}) \quad (6)$$

$$y_k = y_{k-1} + v * h * sin(\theta_{k-1} + \frac{wh}{2}) \quad (7)$$

$$\theta_k = \theta_{k-1} + wh \quad (8)$$

A partir de estas ecuaciones, se pretende obtener el vector de estado X_k. Estas se formulará como aparece en la expresión 9.

$$X_k = \begin{bmatrix} X_{1,k-1} + v * h * cos(X_{3,k-1} + \frac{wh}{2}) \\ X_{2,k-1} + v * h * sin(X_{3,k-1} + \frac{wh}{2}) \\ X_{3,k-1} + wh \end{bmatrix} \quad (9)$$

Con estas expresiones, habríamos obtenido la posición del vehículo estimada por el modelo cinemático y que nos será útil posteriormente para la predicción real de la posición.

3.2 Modelo de observación

En segundo lugar, se va a pasar a desarrollar el modelo de observación del sistema. Este se basa en la detección de balizas en el entorno a partir de un sensor láser modelo "LMS100". De las balizas detectadas se recibirá tanto su distancia, como el ángulo, en el que el haz de puntos láser ha detectado a esta. A partir de esta información, en nuestro caso, se tomará la distancia al láser y las componentes seno y coseno de este ángulo, para mantener la continuidad del ángulo barrido, que luego nos será útil para la comparación de la posición de nuestro vehículo. De esta manera la matriz de observación $Z_{observacion,k}$ queda de como aparece en en la expresión 10.

$$Z_{observacion,k} = \begin{bmatrix} d_{baliza} \\ sin(\alpha_{baliza}) \\ cos(\alpha_{baliza}) \end{bmatrix} \quad (10)$$

siendo d_{baliza} la distancia obtenida del robot a la baliza y ang_{baliza} el ángulo medido del robot a la baliza. De esta manera queda perfectamente definido a las distancias y los ángulos que se encuentra nuestro vehículo de cada una de las balizas. Esto nos servirá posteriormente para corregir los valores dados por la odometría anteriormente calculada.

3.3 Matrices Jacobianas

En último lugar, se debe calcular las matrices Jacobianas como H_k y Φ_k. Para ello el proceso es bastante simple y se detallará en las siguientes líneas.

Para la obtención de H_k se ha tenido que tomar el modelo de observación que se aplica para el sistema cinemático, al que llamaremos $Z_{cinematico,k}$. Este se obtiene con la expresión 11.

$$Z_{cinematico,k} = \begin{bmatrix} \sqrt{\Delta x^2 + \Delta y^2} \\ sin(\Delta\alpha) \\ cos(\Delta\alpha) \end{bmatrix} \quad (11)$$

siendo $\Delta x = x_{baliza} - X_{1,k}$, $\Delta y = y_{baliza} - X_{2,k}$ y $\Delta\alpha = atan(\frac{\Delta y}{\Delta x}) - X_{3,k}$. De esta manera queda definida de forma adecuada la estimación de la posición respecto cada una de las balizas visibles y se puede comparar con la real obtenida del láser.

Tras esto se toma la matriz $Z_{cinematico,k}$ y se calcula su Jacobiano, obteniéndose de esta manera la matriz H_k que se muestra a en la expresión 12.

$$H_k = \begin{bmatrix} -\frac{\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} & -\frac{\Delta y}{\sqrt{\Delta x^2+\Delta y^2}} & 0 \\ \Delta y \frac{cos(-\Delta\alpha)}{\sqrt{\Delta x^2+\Delta y^2}} & -\Delta x \frac{cos(-\Delta\alpha)}{\sqrt{\Delta x^2+\Delta y^2}} & -cos(-\Delta\alpha) \\ \Delta y \frac{sin(-\Delta\alpha)}{\sqrt{\Delta x^2+\Delta y^2}} & -\Delta x \frac{sin(-\Delta\alpha)}{\sqrt{\Delta x^2+\Delta y^2}} & -sin(-\Delta\alpha) \end{bmatrix} \quad (12)$$

Tras la obtención de la matriz H_k, pasaremos a la obtención de la matriz Jacobiana Φ_k. Esta parte de la expresión de X_k calculada anteriormente. Si se le aplica el Jacobiano a esta matriz queda la expresión 13.

$$\Phi_k = \begin{bmatrix} 1 & 0 & -vh * sin(X_{3,k-1} + \frac{wh}{2}) \\ 0 & 1 & vh * cos(X_{3,k-1} + \frac{wh}{2}) \\ 0 & 0 & 1 \end{bmatrix} \quad (13)$$

$$G_k = \begin{bmatrix} cos(X_{3,k-1} + \frac{wh}{2}) & -0{,}5 * vh * sin(X_{3,k-1} + \frac{wh}{2}) \\ sin(X_{3,k-1} + \frac{wh}{2}) & 0{,}5 * vh * cos(X_{3,k-1} + \frac{wh}{2}) \\ 0 & 1 \end{bmatrix} \quad (14)$$

Además se puede ver como aprovechando el cálculo de la matriz Φ_k se ha calculado también la matriz G_k tal y como aparece en la expresión 14.

Es de esta manera como ahora realizando el cálculo de Y_k, mostrada en la expresión 15, como empieza el proceso de comparación y corrección de nuestro Filtro Extendido de Kalman, obteniéndose una estimación final de la posición del robot Marvin en el mapa.

$$Y_k = \begin{bmatrix} Z_{observacion,1,k} - Z_{cinematico,1,k} \\ Z_{observacion,1,k} - Z_{cinematico,1,k} \\ Z_{observacion,1,k} - Z_{cinematico,1,k} \end{bmatrix} \quad (15)$$

4. CONTROL

Para el control del robot se han tenido en cuenta dos situaciones. En primer lugar tenemos el control reactivo, evidentemente, este se encargará de las situaciones que dependan de una reacción por parte de *Marvin* para pasar de un estado a otro siempre que se encuentre en una situación de apuros. Esta situación estará reflejada principalmente en todos aquellos obstáculos que no hayan sido introducidos en el mapeado del entorno como pueden ser papeleras, bancos, farolas o incluso gente que se pueda encontrar a lo largo de su recorrido.

El segundo tipo de control realizado es el llamado de navegación. Este tendrá en cuenta el planificador que comentaremos en el siguiente apartado, tras identificar los *waypoints* o nodos generados que han sido determinados como el camino a seguir *Marvin* tendrá que ir tras ellos con una aproximación definida en el propio control.

Cabe a destacar que el buen funcionamiento del control depende tanto de uno como de otro ya que sin un correcto control reactivo nuestro robot no sería capaz de esquivar objetos y sin un buen control de navegación no sabría como moverse para seguir al planificador pero si que puede ser más susceptible el control reactivo ya que requiere de mayor precisión en las acciones debido a que serán en las situaciones donde este control cobre mayor importancia en las que *Marvin* se pueda quedar atascado.

4.1 Control reactivo

El control reactivo se caracteriza por tener un tiempo de respuesta rápido ya que para su funcionamiento no intercede el mapa sobre el que se trabaja sino el entorno captado por diferentes sensores que sean capaces de apreciar diferentes objetos. Debido a esto, la elección de los sensores encargados de realizar está función es imprescindible que sea adecuada.

Como ya mencionamos en el apartado de sensores se decidió contar con 5 sensores ultrasonidos colocados en la zona frontal del robot. Se plantearon varias variaciones distintas pero tras diversas pruebas y tras la integración completa del sistema se decidió que esta iba a ser la mejor manera para que nuestro robot fallase lo menos posible por la no detección de objetos en puntos muertos o esquinas que se encuentren entre los conos de dos sensores consecutivos.

Como en este proyecto no se tiene en cuenta el coste total del proyecto sino el buen funcionamiento de este consideramos como primordial el cubrir completamente toda la zona frontal y de esta manera la buena detección de estos puntos críticos. Además cabe mencionar que este tipo de sensores son realmente sencillos y baratos.

Con respecto al control reactivo en sí, hemos decidido colocar tres distancias límite en las que se tomarán diferentes valores para velocidad lineal y angular. Estas distancias van a ser 0.6, 0.40 y 0.25 metros que se corresponderán con *no obstáculos cercanos*, *obstáculo no muy cerca* y *obstáculo cerca* respectivamente. Para estas características si el mínimo de los valores tomados por los sensores considerados frontales, (los tres centrales) dan un valor mayor a 0.6 m y los considerados como laterales mayor a 0.45 m estaremos en una situación en la que devolveremos como 1 el control de la velocidad lineal y 0 el control de la angular.

De la misma manera si nos encontramos frontalmente por encima de 0.45 m y lateralmente de 0.25 m devolveremos un valor de 0.2 para la corrección de la velocidad lineal y 0.7 para la velocidad angular, el obstáculo se detecta pero no está cerca. Si los sensores detectan valores superiores a 0.25 m pero, evidentemente, inferiores a 0.45 m el obstáculo está cerca y devolveremos 0.5 tanto para velocidad lineal como angular.

Para todos estos casos se ha tenido en cuenta cual de los dos lados es el que está detectando el valor crítico y en función de eso devolver el valor angular corregido negativamente para que gire en el sentido correcto. Además, el controlador permite dar prioridad a la dirección de navegación en situaciones donde ambas direcciones son válidas.

Finalmente para el control reactivo tenemos la situación de obstáculo realmente cerca, para este caso se devolverá una velocidad de corrección lineal 0 y una angular de 0.5 que esta vez dependa de la dirección que marca el control de navegación. En este caso la velocidad final del robot va a depender única y exclusivamente de esta corrección reactiva como veremos más adelante en la sección 4.3.

4.2 Control de navegación

El control de navegación va a ser el encargado de seguir la ruta generada por el planificador (ver sección 5). Para este control se ha decidido implementar un controlador PID sobre la velocidad angular. En un primer lugar, se calcula el ángulo que debe ser modificado entre los dos *waypoints* siguientes por medio de sus posiciones en el mapa. Tras esto se comparan con el ángulo de la posición estimada, se calculan los errores y se obtiene la próxima velocidad angular.

El error para el controlador PID viene dado por la expresión 16, ya que de esta manera aseguramos la no preferencia por giros hacia un lado, y también, conseguimos continuidad en el error que tomando directamente el error entre los ángulos no conseguiríamos.

$$error = sin(\theta_{waypoint} - \theta_{robot}) + (1 - cos(\theta_{waypoint} - \theta_{robot})) \tag{16}$$

El controlador PID además contará con los siguientes valores de ganancias, que se han ajustado de manera óptima para combinarse de forma adecuada con el control reactivo:

- Kp = 1,00
- Ki = 0,00
- Kd = 0,50

La decisión de mantener la parte derivativa del controlador, es para que sea capaz de adelantarse y reaccionar rápidamente ante cambios grandes de tendencia en la consigna, lo cual es adecuado para nuestra aplicación.

Para la velocidad lineal, si la diferencia que existe entre el ángulo calculado por la localización de los *waypoints* y el ángulo del robot es menor a $\pi/10$, le daremos un valor de 1, y de 0.1 en caso contrario. De esta manera, el controlador intentará primero orientar el robot hacia el siguiente *waypoint* y después avanzar hacia el mismo.

En este punto también se determina si se ha alcanzado el *waypoint* deseado en base a un parámetro de proximidad, cuyo valor habitual es de 30 cm. Esto nos permite saber cuando estamos sobre el objetivo deseado de forma que el controlador no intente ajustar el ángulo estando encima.

4.3 Integración de ambos

Por separado, el funcionamiento de ambos controladores esta bien definido y proporciona resultados buenos. Sin embargo, para poder obtener lo mejor de ambos, es necesario unificarlos en una señal de control única. En algunas situaciones preferimos que el control reactivo tenga más peso, para evitar colisiones, pero hay otras situaciones en que queremos que el control de navegación sea prioritario.

El método para encontrar el mejor compromiso entre ambos ha sido seguir un proceso iterativo ajustando el peso de uno y de otro. De entrada, establecimos que cada una de las señales de control otorgara un valor de 0 a 1, para así poder escalarlo posteriormente. En situaciones nominales, en que no hay obstáculos excesivamente cerca, los comandos de velocidad lineal y angular se calculan según las ecuaciones 17 y 18.

$$v = \frac{(2 * vControl + vReact)}{3} * vMax \tag{17}$$

$$w = \frac{(2 * wControl + wReact)}{3} * wMax \tag{18}$$

En el caso especial, en que un obstáculo esté excesivamente cerca, queremos que la velocidad lineal sea nula, para que el robot no siga avanzando hacia el obstáculo y únicamente gire. En este caso, el control reactivo toma el control absoluto, girando en la dirección libre, priorizando la dirección del *waypoint* siguiente. De esta forma, las consignas de velocidad se obtienen tal y como se muestra en las ecuaciones 19 y 20

$$v = (vReact) * vMax \tag{19}$$

$$w = (wReact) * wMax \tag{20}$$

Se puede observar como en las ecuación intercede tanto el control reactivo (dado por *vReact* y *wReact*) como el control de navegación (dado por *vControl* y *wControl*). Puesto que ambos aportan un valor entre 0 y 1, podemos ponderar ambos de manera sencilla. Por último, se escalan los comandos según los valores máximos establecidos para el robot: $0,25 m/s$ para la velocidad lineal y $1,00 rad/s$ para la angular.

5. PLANIFICACIÓN

La planificación en robótica es la secuencia de acciones que permiten llevar un sistema desde un estado inicial a uno final. Para ello, haremos uso de los diferentes planificadores que existen implementados en *Matlab*, lo integraremos a nuestro programa y será finalmente la directriz que lleve a *Marvin* por la trayectoria encontrada desde el punto inicial al final en principio, libre de colisiones.

Esta trayectoria generada por el planificador tendrá que implementarse en el mapa sobre el que nosotros trabajaremos con el robot y se le tendrán que modificar ciertas especificaciones para que el planificador sea capaz de generar una ruta y a ser posible la ruta más adecuada.

Se ha decidido llevar una planificación por muestreo, en este caso introduciremos dos tipos de planificador por asegurar que va a existir la planificación, en el caso de que nuestro principal falle.

Se ha decidido introducir como principal planificador el RRT, en concreto el RRT*, este va a tener como características principales la amplia exploración del espacio de búsqueda,

no existe la fase de preprocesado, búsqueda diferencial y por lo tanto más difícil la aparición de mínimos locales, trata de igual forma los problemas complejos y sencillos por lo tanto aunque consideré nuestro problema como sencillo lo tratará de forma compleja y tiene crecimiento exponencial con los grados de libertad.

Esto demuestra que es un planificador muy completo. Su funcionamiento se basa en indicarle el punto de inicio y final y con la generación de nuevos vértices a lo largo del mapa se añadirán al punto más cercano ya existente en la trayectoria. Si el vértice generado se encuentra en un obstáculo la línea se generará de la misma manera pero terminará en la intersección entre el obstáculo y la propia línea generada.

En segundo lugar se ha utilizado el planificador PRM básico o planificador probabilístico. Se basa en la generación de un mapa de carreteras en Cfree, generando este de forma probabilística. Los nodos corresponden con configuraciones escogidas de forma probabilística y las aristas entre nodos corresponden con trayectorias sencillas entre dos nodos. Las trayectorias son calculadas por un planificador rápido, de esta manera nos aseguramos que la planificación siempre vaya a salir adelante aunque sea por este segundo método de menor potencia.

El funcionamiento es el siguiente, se generan los nodos de forma probabilística, aquellos que se encuentren en el espacio libre serán los que sigan existiendo, se unen los nodos en línea recta con aquellos más cercanos, aquellas que no atraviesen los obstáculos seguirán existiendo, finalmente se unen el resto de nodos entre si generando un mapa de carreteras que será capaz de llevar al robot del inicio al fin por el camino más rápido buscado en el grafo.

Tras comentar las características principales de los dos planificadores que se van a utilizar en la búsqueda del camino a seguir por Marvin entramos en las condiciones de funcionamiento de ambos. Para ambos planificadores se va a tener en cuenta un radio del robot de 0.15 m, te esta manera ya indicamos que los waypoints tienen que estar como mínimo a esta distancia de los obstáculos que se encuentran en el mapa.

En el caso concreto del RRT* utilizaremos una distancia de validación de 1, que la conexión máxima entre waypoints sea de 1 también y un número máximo de iteraciones de 10000 así tiene suficientes iteraciones para que sea capaz de encontrar una ruta. Para el correcto funcionamiento del planificador sólo nos faltará introducirle la imagen binaria del entorno. Un ejemplo de funcionamiento puede verse en la figura 4.

En el caso del PRM las condiciones iniciales son diferentes, la distancia de conexión entre nodos en este caso será 4, con un número de nodos iniciales de 50 y máximo de nodos totales de 10000, asegurándonos, de la misma manera, que

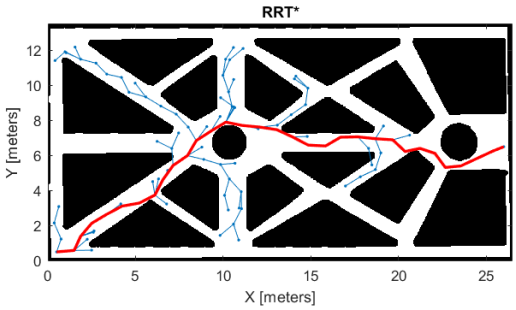

Figura 4. Planificador RRT*

existe una ruta. Este planificador en el caso de que no encuentre ruta de primeras, introducirá 10 nodos más y probará nuevamente con conexiones, así sucesivamente hasta que la encuentre. Un ejemplo de funcionamiento puede verse en la figura 5.

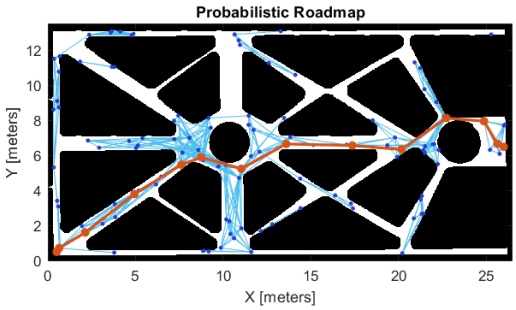

Figura 5. Planificador PRM

6. DEMOSTRADOR

Tras la integración de todos los apartados anteriores se ha llegado prácticamente al final del proyecto. Evidentemente hay detalles que se deben cambiar ya que simulados cada uno por su cuenta si que se tenía un correcto funcionamiento pero una vez unidos, hay puntos claves que no se habían tenido en cuenta. Este puede ser el caso de los controladores, por ejemplo. Individualmente funcionaba cada uno de forma precisa pero una vez integrados se ve como claramente, esquinas u obstáculos que se encuentra demasiado cerca, el control por navegación le manda unas directrices y el reactivo otras.

Por ello el demostrador va a ser clave, ya que es en este punto donde todo lo que se había realizado con anterioridad

se va a equilibrar de tal manera que nuestro robot se mueva correctamente. Es aquí donde se puede apreciar si los sensores elegidos son correctos o hacen falta más o menos, también cuál es el punto concreto en el que se le tiene que empezar a dar más valor al control reactivo conforme nos acercamos a un obstáculo, incluso modificaciones en el propio planificador para adecuar también la ruta al movimiento que estamos viendo en *Marvin*.

El programa de demostración [1] integra todo lo descrito anteriormente en un solo programa, lo que nos permite poner en funcionamiento a *Marvin* en el entorno simulado. La secuencia de ejecución es la siguiente:

1. Configuración e Inicialización
2. Planificación
3. Bucle de funcionamiento
    a) Localización
    b) Replanificación (si fuera necesario)
    c) Selección de *waypoint* (si fuera necesario)
    d) Control de navegación
    e) Control reactivo
    f) Computar velocidad lineal y angular en base a los controladores
    g) Comandar movimiento y sincronizar
4. Mostrar resultados

Así pues, en una ejecución, se puede configurar un punto del mapa como objetivo y el robot intentará generar un camino válido a través del mapa y seguirlo evitando colisiones con obstáculos ya sean conocidos o no.

6.1 Configuración e inicialización

El demostrador permite establecer algunos datos que condicionan la ejecución del programa. Algunos de los más relevantes permiten modificar el punto inicial del robot, la ubicación objetivo, los parámetros del controlador, variables del planificador y velocidades máximas. De esta forma podemos ajustar el comportamiento de *Marvin* durante la ejecución y los objetivos a seguir.

El siguiente paso es inicializar todas las variables y estructuras con valores de inicialización. Esto supone una parte necesaria para que el programa pueda funcionar correctamente y garantiza que todos los datos que se van a utilizar estén instanciados y con valores controlados.

[1] `https://youtu.be/pEcNK_Qjzcw`

6.2 Planificación

Antes de empezar el movimiento del robot, el demostrador genera un camino válido que lleve desde el punto de inicial al objetivo usando un mapa del entorno. Para ello hace uso de los planificadores descritos en las sección 5. De esa manera, se intenta generar un comino usando el *RRT**, que en caso de no ser capaz de hallar el camino entre ambos puntos, se recurriría al *PRM*. De esta formas tenemos la garantía de que vamos a encontrar un camino válido.

El planificador nos devuelve una secuencia de puntos en el mapa que permiten llevar al robot por el entorno sin encontrar obstáculos, siempre y cuando esos obstáculos estén representados en el mapa.

6.3 Bucle de funcionamiento

Aquí el demostrador entra en el bucle principal del programa donde se va a llevar el proceso iterativo para localizar y posicionar *Marvin* en el entorno. Consiste básicamente en un bucle *while* que termina si el robot colisiona, si se termina el tiempo máximo de ejecución o si se llega al objetivo. El bucle realiza una serie de pasos en cada iteración:

Localización: Lo primero que se hace en cada ciclo es localizar el robot en el entorno. Para ello se hace uso del *Filtro de Kalman Extendido*, explicado en detalle en la sección 3. Realizamos esta operación en primer lugar, puesto que esto nos permite contar con la mejor aproximación posible de la posición del robot para necesidades posteriores.

Replanificación: En el caso de que *Marvin* se desvíe de la ruta establecida por el planificador, ya sea por un problema en alguna de las partes o por un obstáculo no esperado, es necesario generar un nuevo camino que permita retomar la ruta hacia el punto objetivo. Para decidir si es necesario replanificar la ruta, se establece un radio que depende de la distancia al siguiente *waypoint*. Este radio se reduce en cada iteración en un valor proporcional a la velocidad máxima que se le permite alcanzar a Marvin y se restablece cada vez que se cambia de *waypoint*. Si la distancia al siguiente punto de ruta es mayor que el radio de replanificación, entonces se genera la nueva ruta de igual manera a como se hizo en primera instancia y se pasa al siguiente ciclo del bucle.

Selección de *waypoint*: A continuación se selecciona el siguiente punto de ruta de la secuencia generada por el planificador. El demostrador permite seguir más o menos fielmente una ruta utilizando un parámetro de configuración. Este parámetro define un radio entorno a *Marvin* de forma que si el siguiente *waypoint* se encuentra dentro de dicho radio, se elegirá el siguiente. En este punto, al cambiar de punto de ruta, también se recalcula un nuevo radio de replanificación mencionado anteriormente.

Control de Navegación: Ya contamos con el punto en el que estamos gracias a la localización y hemos definido el siguiente punto de ruta a seguir. Con estos datos, ejecutamos el

controlador encargado de la navegación. Los detalles de cómo funciona se pueden consultar en la sección 4.2. El controlador toma como entrada los parámetros configurados previamente así como las posiciones del robot y del siguiente *waypoint* para devolver la velocidad lineal y angular necesaria para seguir la ruta.

Control Reactivo: En el siguiente punto, se ejecuta el controlador encargado del manejo reactivo del robot. Los detalles pueden consultarse en la sección 4.1. Este controlador evalúa la cercanía de los obstáculos cercanos haciendo uso de los sensores de ultrasonidos y genera una velocidad lineal y angular con el objetivo de mantenerse lo más lejos posible de ellos. Además también toma como entrada una dirección preferente hacia la que girar, que permite coordinarlo con el control de navegación para que se intenten esquivar los obstáculos siguiendo la trayectoria. Por último, el control reactivo también presenta un caso especial en el que la colisión es inminente, en la cual se sobre escribe el control de navegación por completo para esquivar el obstáculo.

Computo final de velocidades: Con las consignas de ambos controladores en mano, el siguiente paso es generar las velocidades tanto lineal como de giro que se le van a comandar a *Marvin*. Cómo se hace esto exactamente puede consultarse en la sección 4.3.

Comandar movimiento y sincronizar Por último, para cerrar el ciclo se envía el comando de movimiento a *Apolo* usando la función *apoloMoveMRobot* pasándole como parámetros las velocidades computadas y el periodo de cada ciclo. También se hace una pausa y una actualización de *Apolo* para que se muestre la evolución en el simulador de manera realista.

6.4 Mostrar resultados

Lo último que realiza el programa demostrador es recabar los datos obtenidos durante la simulación y presentarlos gráficamente. En primer lugar se muestra como ha evolucionado el vector de estados del robot a lo largo de la simulación, tal y como se observa en la figura 6. Esto incluye la trayectoria real que ha seguido el robot y las estimaciones que ha ido realizando en cada iteración.

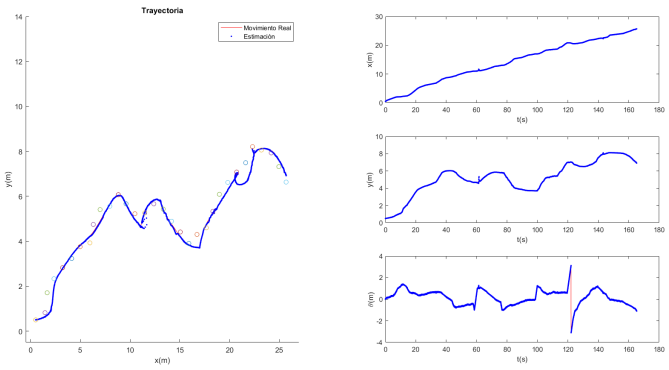

Figura 6. Resultados de localización

Por otro lado se muestra el mapa del entorno con la trayectoria que ha seguido el robot por el jardín así como los *waypoints* que debía seguir y las ubicaciones de las balizas. Un ejemplo de la ejecución puede verse en la figura 7.

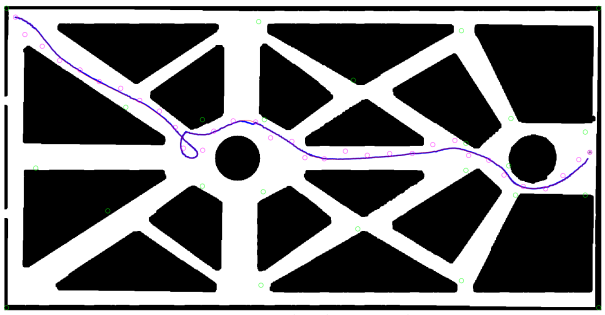

Figura 7. Resultados en el mapa

Previamente, al concluir la planificación, también se muestra el recorrido a seguir en una imagen, tal y como se puede ver en las figuras 4 y 5. Con esto podemos comparar con la trayectoria del robot para ver cuán fielmente se ha seguido.

7. CONCLUSIÓN

Este proyecto tenía como objetivo principal la familiarización de los temas principales que aparecen en la navegación de un robot en presencia de incertidumbre: plantear el problema definiendo el mapa del entorno sobre el que se va a trabajar, identificar el sistema de locomoción, escoger el sistema de percepción, entender y programar el algoritmo de localización, realizar el control del vehículo y trabajar con un planificador capaz de generar trayectorias en presencia de obstáculos. Finalmente preparar un demostrador en el que se mostrase la integración de todos los puntos anteriores.

El resultado es claramente satisfactorio, se han cumplido todos los objetivos, tratando, a la vez, de introducirnos en este ámbito con un proyecto original no solo en el problema a resolver sino también en la solución propuesta. Es una idea que hoy en día no se ve implementada en la sociedad pero que perfectamente puede aparecer en los años próximos si sigue esta tendencia de introducir cada vez más la robótica no sólo para su uso industrial sino también para su aplicación social. De la misma manera se ha resuelto el problema de forma precisa y clara, siguiendo una evolución ascendente y no solamente arreglando los errores que iban surgiendo sino mejorando el propio movimiento del robot y todo el sistema que lo rodea.

En cuanto a la gestión se refiere, el grupo ha sabido trabajar en equipo con una buena coordinación, realizando el trabajo gradualmente y reuniéndose aproximadamente cada dos semanas con el fin de ordenar las ideas, sacar puntos en común y cuando el trabajo ya estaba más adelantado, avanzarlo conjuntamente tratando de entender los errores que podíamos

estar teniendo y planteando el como poder solucionarlos. Todo esto siempre en un marco de trabajo totalmente remoto debido a la situación actual, con las complicaciones que esto conlleva. Aún así, se ha sabido sacar provecho de las herramientas que tenemos a nuestra disposición, tales como *Teams* y *Whatsapp* para las comunicaciones, *Github* para el control de versiones y el desarrollo conjunto del software y *Overleaf* como plataforma para el desarrollo de la documentación.

Desde un punto de vista crítico, somos conscientes que el resultado no es perfecto, el control reactivo, por ejemplo, no es exacto, funciona adecuadamente y se ha logrado integrar con el control de navegación con buen resultado como se ha redactado a lo largo del documento, pero si que es cierto que de vez en cuando genera giros innecesarios para esquivar obstáculos en la dirección contraria a la que tendría que moverse y también, a veces, no es capaz de reaccionar correctamente con las esquinas, principalmente debido a la incapacidad de los ultrasonidos de detectarlas correctamente. Sobre el resto de puntos, tal vez exista margen de mejora pero, como se ha mencionado, estamos satisfechos con el trabajo realizado.

Entre las líneas futuras que se pueden contemplar, principalmente estaría darle más forma a la funcionalidad que se ha planteado para el proyecto. En principio nuestro robot es capaz de moverse libremente por el entorno siendo capaz de esquivar obstáculos y llegar al punto que se le pida, pero se puede observar como la trayectoria generada no es descrita por líneas rectas cercanas a los arbustos (la manera más correcta de que el robot pueda realizar su función de jardinero), por lo tanto, este sería un punto clave a tratar. También, en esta misma perspectiva, haría falta añadir las herramientas que *Marvin* necesitase, así como la propia programación de su funcionamiento para poderlo montar en el hardware real. Por último, quedaría integrar todo lo desarrollado en una aplicación funcional con una interfaz de usuario amigable que permitiera a usuarios no especializados hacer uso de esta tecnología de cara a comercializarlo como robot ayudante en la poda de setos de grandes jardines.

Para concluir, consideramos que el proyecto no solo ha sido muy interesante, sino que el aprendizaje que venía detrás era, además de necesario, muy enriquecedor. A pesar de ser un trabajo extenso con mucha dedicación, claramente, este es uno de esos casos en los que el esfuerzo tiene recompensa y por ello estamos contentos tanto con el trabajo realizado como con los resultados finales obtenidos.