# Diseño e implementación de un algoritmo de guiado óptimo para superficies devastadas o catastróficas en robots autónomos

Autor: Alejandro Morón Mas

Profesor: Federico José Barrero García

**Dpto. Ingeniería Electrónica**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2020

# ÍNDICE

# 1. Introducción

La progresiva implantación de la robótica en nuestras vidas anuncia importantes cambios, cosa que no siempre conlleva la sustitución de los trabajos realizados tradicionalmente por el ser humano, como algunos temen, aunque sí obligará a una redistribución de las tareas en aras de la eficacia y la eficiencia. Entre los diversos tipos de robots destacan los de uso industrial, médico, espacial o militar. Son especialmente útiles para la realización de tareas rutinarias, pero también, como se explicará a continuación, para las más difíciles y arriesgadas.

El campo de la ingeniería robótica lleva varios años tratando de diseñar máquinas capaces de suplantarnos en esos trabajos. Existen muchos ejemplos de su aplicación, son capaces de recorrer una zona catastrófica o radiactiva, comprobando a su vez las causas que lo provocaron. También pueden ofrecer apoyo médico a personas aisladas en zonas peligrosas o de difícil acceso para el ser humano, o incluso desactivar explosivos sin poner en riesgo vidas humanas. Por ello, la finalidad de este Trabajo de Fin de Grado es ayudar a la realización de esos trabajos arriesgados pero necesarios, mediante el diseño de un vehículo autónomo que sea capaz, no solo de atravesar zonas catastróficas o inhóspitas, sino también de realizar diversas tareas, como las propuestas anteriormente. Para demostrar su fiabilidad, se implementará el algoritmo desarrollado en este trabajo en el microcontrolador de Texas Instruments TMS320F28335.

En esta memoria se recoge el estado en el que se encuentra el arte del diseño de robots autónomos, junto con las tecnologías y algoritmos predominantes en el sector. Se centrará en el mapeo y sus problemas, es decir, en la capacidad de construir un mapa basándose solamente en los datos que registran los sensores de nuestro robot autónomo. También se tratará, la compleja tarea de localización de éste, ya que, será necesario facilitarle varios datos odométricos al autómata. La odometría sirve para estimar la posición relativa del robot respecto a su localización inicial.

Uno de los principales temas a tratar en estas páginas será el tratamiento de errores de cualquier tipo, en concreto trataremos uno de los problemas más frecuentes que se presentan en la robótica actual, aplicar la técnica del "*Simultaneous Location And Mapping*", en adelante SLAM. Se trata de una técnica que permite a los vehículos autónomos y a los robots detectar su entorno y construir, a su vez, un mapa fiable para poder desplazarse en él de forma precisa y segura, construyendo su trayectoria. Esta técnica es el resultado de los esfuerzos que mantiene la robótica desde hace más de veinte años intentando perfeccionar la navegación autónoma de los robots.

Asimismo, analizaremos el estudio y las distintas formas de abordar el problema, a través de métodos bioinspirados o probabilísticos.

Los métodos bioinspirados, como su propio nombre indica, consisten en el uso de algoritmos bio-inspirados. Existe una rama de una rama de la Inteligencia Artificial, en adelante IA, que se dedica a su estudio. El método, trata de conseguir emular el comportamiento biológico del entorno, con todas las posibles variables que se pueden presentar.

En cambio, los métodos probabilísticos, tratan de resolver los problemas que generan las fuentes de incertidumbre implicadas en el proceso, mediante algoritmos, diseñados a partir de datos que se conocen. Posteriormente, se analizan los distintos métodos de procesado de datos que la ciencia ha puesto a nuestra disposición para poder resolver este problema analizando las ventajas y desventajas de cada uno de ellos, en concreto mediante los siguientes métodos:

- Filtros de Kalman
- Filtros de Partículas
- Grafos

Una vez planteado el problema SLAM, profundizaremos sobre otro distinto en la navegación, la búsqueda del camino más corto o camino óptimo. Este objetivo, propone, dada una superficie, ser capaz de encontrar el camino más corto evitando todo tipo de obstáculos que se planteen. La finalidad será detectar las distintas variables que pueden aparecer para ir así solventando las dificultades que se presenten.

Para resolver este problema varios matemáticos optaron por reducir los mapas a grafos o mapas topológicos, que, como se explicará posteriormente, no son más que representaciones de un mapa que han sido simplificadas de tal modo que con el mínimo peso se pueda transmitir la información básica y necesaria para que nuestro sistema pueda funcionar correctamente.

Analizaremos distintos modelos, aunque el foco del estudio se centrará en uno, el algoritmo de Dijkstra. Este algoritmo es de carácter recurrente y muy optimizado, aunque no puede ser usado en todo el espectro de grafos, ya que se parece en gran medida al planteamiento que desarrollaremos en este trabajo.

El SLAM visual, no es objeto de estudio en este Trabajo de Fin de Grado (en adelante TGF), aunque debido a su importancia, sí es justo señalar los avances que están habiendo en este campo. Está basado en que la información recibida, se percibe a través de una cámara que funcionará como los ojos del robot, procesando así la imagen recibida y abordando el problema desde otra perspectiva. El motivo de la exclusión de su análisis en el presente trabajo es que este tipo de SLAM, está en un tramo pseudoinicial, además, en este TFG no se trabaja con ningún tipo de procesado de imagen.

Después del análisis exhaustivo del estado del arte de la navegación robótica, se procede al análisis del hardware del coche. Se plantea el estudio sobre el microcontrolador que lo maneja y sus más importantes ventajas; las ruedas y la disposición y conexión con todos los periféricos que conforman el vehículo autónomo.

Se describirán las conexiones básicas del robot que hacen posible que éste funcione, basándonos en diagramas de bloques y distintas imágenes para aportar mayor claridad a la metodología de desarrollo del hardware.

Este algoritmo se basa, como se ha dicho anteriormente, en el algoritmo de Dijkstra, también denominado algoritmo de caminos mínimos que se incluye dentro de los algoritmos de búsqueda. Su objetivo principal es en encontrar la ruta más corta desde el nodo de origen hasta cualquier otro nodo al que se pretenda llegar.

Este algoritmo será ligeramente modificado en el sentido de no basar su funcionamiento en grafos, sino que el código recibirá una matriz de datos y se encargará de crear el camino más corto de una punta a otra del mapa.

Este sistema tiene un uso polivalente, pudiéndose utilizar, por ejemplo, para recorrer superficies de zonas catastróficas. En numerosas ocasiones, serán de difícil acceso, pero resultará crucial poder enviar material o medicinas de un punto a otro para salvar vidas humanas. El acceso que puede ser complicado o casi imposible para un humano puede ser factible para un robot. En este momento entra en juego la necesidad de conseguir que el robot,

no sólo para desplazarse de un punto a otro, sino que lo haga en el menor tiempo posible, ya que el tiempo puede ser crucial.

Una posible forma de trabajar con el robot sería que un dron hiciera una foto aérea de la superficie que el sistema ha de recorrer. Posteriormente se procesa para que el robot elabore la ruta más eficiente consiguiendo así su objetivo. Sin embargo, para dotarle de autonomía se ha decidido realizar un mapeo previo de la superficie a recorrer.

Cabe decir que el algoritmo no se queda simplemente en recorrer una matriz y optimizar el recorrido, una vez hallado el camino el robot es capaz de recorrer la superficie dividida en cuadrados, estos pueden estar ocupados o no. Para ello se muestra cómo se implementa el código y se presentan todas las variables del proceso que se asocian a las acciones. Posteriormente, se recogen una serie de resultados en distintas situaciones y con distinta disposición de obstáculos, para así mostrar la robustez y fiabilidad del código propuesto.

Basándonos en estos resultados y las conclusiones que se sacan del estado del arte, podremos elaborar una fundada opinión sobre el uso y la utilidad de este sistema de procesado comparado con los otros sistemas que existen en este ámbito, proponiendo así, una nueva perspectiva de cómo abordar los problemas de navegación robótica.

# 2. Estado del arte

## a) Robot Autónomo. Definición

Etimológicamente hablando la palabra *robot,* usada internacionalmente, es de origen checo, significa "trabajos forzados". Por otro lado, existe una acepción popular de lo que se entiende por robot, influenciada principalmente por los libros y el cine. La imagen estereotipada de un robot es un androide, es decir que tiene forma humana, obviando que pueden ser de todas formas y tamaños; pequeños, grandes, redondos, cuadrados o con forma de insecto entre otros. Lo importante es que se trata de máquinas complejas programadas para hacer trabajos, que también pueden hacer música, o jugar al fútbol.

Los robots se pueden utilizar para construir y ensamblar objetos, por si solos o tele operados o controlados remotamente, pudiendo hacer más de un tipo de trabajo.

En cuanto a la definición de robot autónomo, en el campo de la ingeniería, no existe una respuesta consensuada, tal vez porque hay tantas definiciones como tipos de robots.

Consultando el diccionario, la primera acepción que la RAE recoge sobre el término robot es: "máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones". Partiendo de esta definición, se infiere, que el robot utilizado en este TFG es el medio en el que se aplicará la IA para conseguir el objetivo propuesto. [1]

Por otra parte, la NASA explica, *"…La robótica es el estudio de los robots. Los robots son máquinas que se pueden utilizar para hacer trabajos. Algunos robots pueden hacer el trabajo por sí mismos. Otros robots siempre deben tener una persona diciéndole qué hacer…"(This article is part of the NASA Knows! (Grades 5-8) series).*

No podemos encontrar, por tanto, un concepto de robot autónomo, aunque sin embargo, es palmario y no se discute que está estrechamente vinculada con la IA. Peter Norvig y Stuart Russell, expertos en ciencias de la computación la definen indicando que *Artificial Intelligence is the study of agents that exist in an environment and perceive and act*. [2] Que se traduce como, la IA es el estudio de todos los agentes que existen en un entorno, y tiene como finalidad percibirlos y actuar en consecuencia.

Es decir, su estudio tiene como finalidad de conseguir que las máquinas repliquen el comportamiento del ser humano en diversos aspectos como deducción, adaptación o razonamiento lógico. Ejemplos de esto pueden ser los asistentes personales de los smartphones como Siri (Apple) o Bixby (Samsung).

Por tanto, con esta definición que se obtiene de la IA, se puede avanzar hasta la de robot autónomo. Cuando hablamos de algo autónomo nos referimos a algo que sea capaz de actuar por sí mismo con cierta inteligencia y adaptación, es decir, replicando a un humano. De esta forma, para que el robot actúe con autonomía, en realidad lo que se pretende es que se aplique en su diseño la IA de forma que pueda conseguir el objetivo requerido. [3}

A raíz de lo anteriormente expuesto se puede entender como robot autónomo, aquella máquina programable que será capaz de realizar distintas operaciones basándose en la metodología de una IA, observando su entorno y actuando en consecuencia para lograr resolver el problema planteado.

## b) Problema SLAM. Mapeado y Tracking

Una vez definido el concepto de robot autónomo ahora el estudio se centrará en el mapeado, el seguimiento y finalmente el mapeado con seguimiento simultáneo o el guiado óptimo.

El profesor Cyrill Stachniss de la Universidad de Friburgo, en su trabajo *introducción* al *mapeo de robots*, explica qué es el "*Simultaneous location and mapping*", en adelante SLAM, a la vez que plantea su principal problema. Explica que hay que localizar al robot y mapear el entorno al mismo tiempo.

Sin embargo, para resolver todos estos problemas existe un obstáculo básico, el robot no sabe ni dónde se halla ni cómo es la superficie en la que se encuentra. Por ello, el robot para ser capaz de trasladarse de manera autónoma, aunque esté situado en un entorno desconocido, debe ir construyendo, de manera incremental, un mapa fiable del ambiente mientras que determina su posición dentro de este mapa. Hallar una solución efectiva a este problema es uno de los principales objetivos todavía pendientes de la comunidad científica Cuando se alcance su solución los robots tendrían las herramientas necesarias para ser totalmente autónomos sin importar el terreno.


Imagen NASA

Este problema y su solución adquieren gran relevancia en situaciones críticas como la exploración espacial, el desplazamiento a través de superficies catastróficas o devastadas o el movimiento de coches autónomos como el de Google. El robot puede saber dónde se encuentra y qué tiene a su alrededor. Esto es así porque sus sensores le informan, aunque no siempre de forma precisa, generando una incertidumbre considerable. El inconveniente surge cuando se producen situaciones que alteran la percepción de los mismos, como el ruido ambiente, un movimiento impreciso del robot, un entorno dinámico u observabilidad parcial.

Figura 1. Rover de Marte.

Antes de abordar los enfoques de resolución típicos para el problema SLAM es necesario clasificarlos, pues no todos los problemas SLAM son iguales ni tienen el mismo objetivo final. Existen varias distinciones, entre las que podemos destacar:

- **Online SLAM versus Full SLAM:**

El problema de *online SLAM* se basa en que nuestra IA solamente tendrá en cuenta los últimos resultados obtenidos, dando así prioridad a la última marca que ha observado con anterioridad.

Se fundamenta en la condición del teorema de la manta de Markov, el cual enuncia que en un sistema basado en estados la variable $\mathbf{X}_t$ dependerá sólo de la variable anterior $\mathbf{X}_{t-1}$, que a su vez dependerá de la anterior, y así sucesivamente. Por tanto, este teorema en sistemas que dependen del tiempo, libera la necesidad de tener que almacenar todas las variables de estado de nuestro sistema, ya que sólo debemos tener en cuenta el estado anterior.

Por otro lado, se presenta el problema de *Full SLAM*, según el cual, hay que tener en cuenta todas y cada una de las variables de estado que se manejan en nuestro sistema para dar así una resolución completa a la superficie en la que estamos trabajando. Por esta razón, el planteamiento resolutivo que se plantea con mayor asiduidad es el de *Graph-SLAM* o SLAM basado en grafos, que se tratará en páginas posteriores.

- **Paramétrico versus no paramétrico**

En el mundo se pueden dar distintos tipos de situaciones que involucren a nuestros sistemas, normalmente, para facilitar el desarrollo de un teorema o teoría se suelen suponer ciertas condiciones que, pueden o no terminar ocasionándose.

En nuestro entorno no todas las distribuciones de probabilidad son parametrizables, de hecho, hay muy pocas. La distribución gaussiana, por ejemplo, es una de esas pocas que son parametrizables, por eso el Filtro de Kalman, algoritmo que estima el estado de un sistema a partir de datos medidos, basa el desarrollo de todos sus sistemas en que funcionan con una distribución gaussiana. Esto por tanto queda limitado para ampliar su efecto.

Sin embargo, existen otros métodos de resolución, como, por ejemplo, el del filtro de partículas, que veremos más adelante, el cual no depende de ninguna distribución probabilística parametrizada.

- **Topológico versus métrico**

Cuando hablamos de SLAM topológicos o métricos en realidad hablamos de su manera de afrontar el desarrollo de un mapa y su avance por él.

En caso de mapas métricos, la explicación es obvia, son mapas cuyas formas de detectar la posición y los obstáculos se basan en la definición de distancias, ya sean absolutas o relativas. Esta solución parece bastante sencilla, sin embargo, puesto que en el problema de SLAM nosotros tratamos de localizarnos a la vez que mapeamos y nos movemos pueden ocurrir ciertas discrepancias con respecto a lo esperado, ya que debido al ruido de sensores o actuadores cambie la situación supuesta y nuestro robot realice un mapa erróneo, o peor, en caso de estar en funcionamiento en servicio de seres humanos, éste provoque algún tipo de desastre o simplemente se rompa.

De esta forma, como alternativa a estos mapas métricos surge la idea de un mapa topológico. Estos mapas están muy simplificados, de forma que nos den solamente la información necesaria para la tarea que debemos hacer. Un ejemplo de este tipo de mapas es el típico mapa de conexiones de metro de grandes ciudades, como, por ejemplo, de Londres, como en la figura 2.

El motivo por el cual estos mapas son tan usados es simple, al estar tan resumidos su peso es mínimo, pero su contenido es máximo de forma que su uso está bastante optimizado. La forma de que un robot entienda esos mapas es simple. Se expresará a través de un grafo en el cual los nodos serán nuestros obstáculos o landmarks y las aristas representarán la información necesaria para llegar a ese punto.


*Fig. 2 Mapa topológico del metro de Londres*

- Estático versus dinámico

En la gran mayoría de casos y por sencillez en los cálculos, se supone que el entorno en el que se basa nuestro sistema es estático, es decir, que no se mueve. Sin embargo, en los últimos tiempos se ha desarrollado también un avance en los estudios del problema SLAM donde el entorno es dinámico, esto provoca bastante mayor coste computacional y un tiempo de ejecución mucho menor, puesto que los objetos pueden aparecer sin que el robot los perciba en un primer momento. [4]

Una vez comentado esto, trataremos dos enfoques de resolución para este problema, los métodos bioinspirados y los métodos probabilísticos:

## c) Métodos bioinspirados

Este primer método se basa en resolver el problema del SLAM por medio de un enfoque biológico, es decir, resolviendo la pregunta de cómo ciertos animales son capaces de guiarse por zonas sin necesidad de la vista. Para este método una amplia mayoría del espectro científico se ha basado en el estudio del comportamiento de las ratas, tras sus experimentos han concluido que todo el funcionamiento del problema a abordar está localizado en el hipocampo del animal, y con mayor exactitud en la corteza entorrinal.

El hipocampo es una estructura que se halla en el cerebro de todos los seres vivos, y tiene una gran relevancia en aspectos de aprendizaje, codificación de memoria y navegación espacial. Por otro lado, la corteza entorrinal se halla en el lóbulo temporal medio del cerebro y se encarga de la conexión entre el hipocampo y el neocórtex, de esta forma podemos apreciar que estamos en el lugar adecuado en el que realizar nuestros experimentos.[7]

Tras todos los estudios realizados, los científicos encargados han encontrado varias respuestas del porqué son capaces las ratas de afrontar el desafío de la navegación espacial, basadas principalmente en dos tipos de células, las células de lugar (Place cells) y las células de red (grid cells).

- **Células de lugar**

Las células de lugar no son más que unas células programadas por nuestro cerebro para activarse en caso de encontrarnos en una posición particular. A medida que un animal se desenvuelve por su entorno estas células se pueden activar en caso de que haya un obstáculo, como consecuencia se activará también el campo de lugar, asociado a dicha célula. Este campo de lugar es la manifestación en el cerebro de nuestro animal, en este caso la rata, con forma de punto caliente, es decir, a menor distancia de nuestro obstáculo mayor será el disparo neuronal, si se halla a mayor distancia éste será menor. Cabe decir, que si esa zona no ha sido explorada por el roedor esta célula no provocará un disparo, ya que la rata construye el mapa basándose en su propia experiencia, de la misma forma en la que se plantea resolver el problema del SLAM.

Sin embargo, las formas de estos campos de lugar no son siempre circulares, si por ejemplo nos encontramos como obstáculo una pared la forma de este campo no será redonda, sino que se adaptará a la superficie que supone una interrupción al camino. Otro punto a tener en cuenta es que estos mapas que crea el elemento de estudio en su entorno son relativos, es decir si giramos cualquiera de las dos imágenes anteriores veremos que se encuentra perdido, sin embargo, una vez que localiza el primer punto caliente su cerebro es capaz de completar el mapa si el entorno simplemente ha girado o se ha desplazado, pero al cambiar objetos de su posición inicial perderá la proporcionalidad de las distancias y tendrá que volver a situarse en el entorno escogido

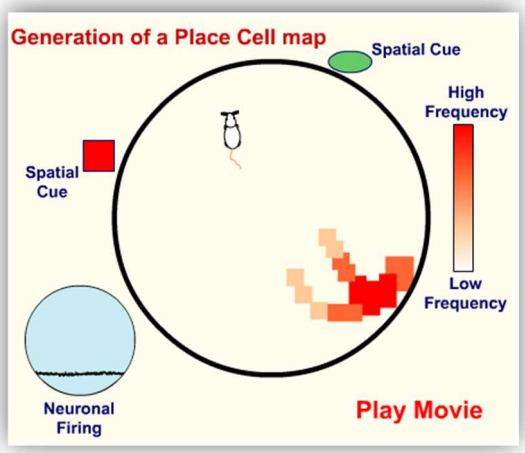



Figura 3. Nuestra rata se halla alejada de nuestro obstáculo, disparo neuronal casi nulo.

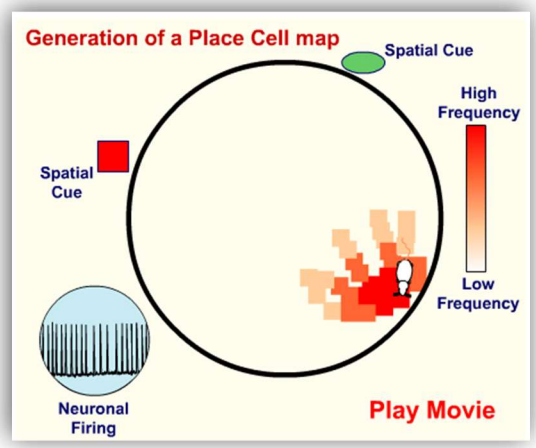


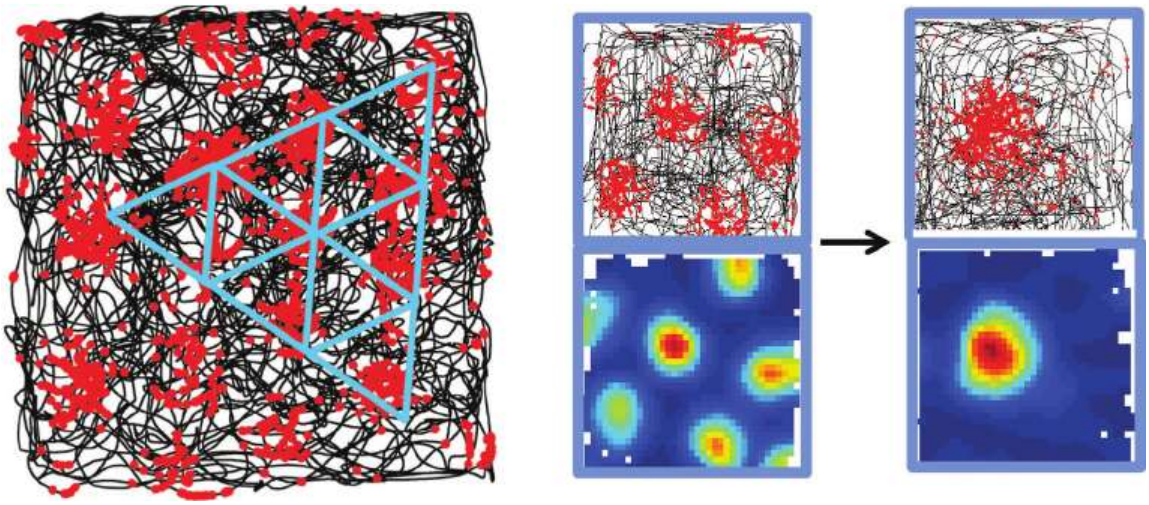

Figura 4. La rata se halla al lado del obstáculo, disparo neuronal máximo

Estas células, obviamente, no son infinitas y de ahí surge la duda de hasta dónde son capaces de memorizar nuestros sujetos de estudio las zonas a la que han sido expuestos. Tras varios estudios, se concluyó que estas células se pueden adaptar al entorno y sabiendo en cuál se encuentran disparar uno u otro campo de lugar, es decir, cada célula es capaz de almacenar más de un sitio en el que el animal se haya encontrado y dependiendo de donde se encuentre cogerá una información u otra. Este descubrimiento resuelve la duda planteada anteriormente y nos aporta la información de que la capacidad de las ratas de recordar y mapear sitios es mucho mayor de lo que nos esperábamos. A este hecho se le ha dado el nombre de remapeo. [6]

- **Células de red**

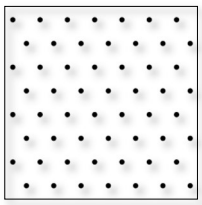
Las células de red, aunque similares a las anteriores, no son las mismas. Éstas se encargan de codificar una representación del espacio euclídeo, es decir, a medida que la rata va avanzando, se puede comprobar que existen distintos puntos de la superficie en los que algunas células de la corteza entorrinal mostraban picos de actividad, pero lo más interesante no era eso, sino el patrón que seguían.

Tras varias comprobaciones para asegurarse de que ese pico no era mera casualidad, los científicos encargados del experimento demostraron que el patrón de activación de esas células,



que de ahí en adelante llamaron “grid cells” (células de red), era de forma hexagonal, así, ese patrón era capaz de constituir un mapa euclídeo de la superficie y junto con las células de lugar era capaz de crear un mapeo completo de la superficie en la que se encontraba, es decir, un GPS en el cerebro.
Figura 5. Izquierda y centro, células de red y su patrón. A la derecha, células de lugar y patrón

De esta forma a raíz de la última imagen podemos distinguir las propiedades de ambas células, se puede comprobar que nuestras células de red conforman un entramado hexagonal disperso por todo el ambiente, mientras que las de lugar no tienen por qué. Además, nuestros puntos de activación normalmente serán equidistantes y estarán rodeados de 6 puntos más entre los cuales habrá un ángulo de 60 grados de separación entre uno y otro.



6. Entramado hexagonal teórico

Cabe destacar que el estadounidense John O'Keefe y el matrimonio noruego formado por May Britt Moser y Edvard I. Moser recibieron el galardón del Nobel de Medicina en 2014. Fueron los científicos que llevaron a cabo el estudio y descubrimiento de las células de red y lugar. [8]

A raíz de lo expuesto anteriormente, se ha elaborado un modelo de resolución del problema SLAM, y se ha hecho basando su funcionamiento en el comportamiento, en mayor medida, de las células de lugar y de las de red. A esta teoría se la ha denominado como RatSLAM.

Esta metodología, se basará así en el funcionamiento del hipocampo de una rata y tratará de replicar por medio de sensores y actuadores el comportamiento de la corteza entorrinal de la rata. Almacenará en memoria las posiciones marcadas tanto por las falsas células de lugar como las de red, para así conseguir el mapa completo de la superficie a abordar. Existe una amplia confianza en esta metodología de resolución del problema SLAM, y sus principales propulsores, como Michael Midford o Adam Jacobson, afirman que esta tecnología será capaz de adelantar a la basada en métodos probabilísticos. [9]

Para finalizar el apartado de los métodos bioinspirados, es necesario hablar previamente de la neurorobótica. Esta disciplina de la ciencia, como su propio nombre indica, trata de unir la neurología, o ciencia cuyo estudio se basa en las neuronas y su funcionamiento, con la robótica o inteligencia artificial. Pretende basarse en el funcionamiento de transmisión de nuestras neuronas y conexiones sinápticas para replicarlas en el funcionamiento de un robot y así resolver cualquier problema que nos interese.

Obviamente el problema del RatSLAM está claramente basado en estudios de neurorobótica, sin embargo, esta disciplina alcanza a más sectores como:

- Control de motores, se están creando motores capaces de adaptarse rápidamente al cambio por medio de metodologías de aprendizaje, en las cuales uno mismo enseña al robot como debe moverse y él lo irá replicando hasta alcanzar la perfección. Un ejemplo

de esto pueden ser los brazos robóticos que se están creando de forma que ellos mismos por medio de la metodología de acierto-error entienden qué deben hacer.

- Relaciones interpersonales, se intenta, por decirlo de alguna forma, que el robot desarrolle sentimientos, y por medio de diferentes estudios se ha conseguido que un robot reaccione de forma negativa o positiva a un estímulo.

- Sensado, fueron de los primeros sistemas neurorobóticos y se basan en conseguir que el robot tome conciencia de su entorno a través de distintos sensores, el más habitual, la vista, sin embargo, también existen estudios en los que se intentan tomar medidas auditivas.

## d) Métodos probabilísticos

Esta metodología para afrontar el problema SLAM fue la primera en ser tomada en cuenta y aportó grandes avances a esta rama. La causa es sencilla de entender, el estudio de la probabilidad es bastante anterior al de la neurociencia y se encontraba bastante más consolidado dentro de la comunidad científica. El estudio para afrontar el problema SLAM fue por primera vez propuesto en 1980 donde distintos científicos por medio de distintas metodologías dieron una posible solución a este desafío, sin embargo, aunque diferían en el método final para obtener los datos necesarios, todos tenían algo en común, se basaban en un inicio en las redes bayesianas. [10]

Thomas Bayes era un matemático británico que vivió en el siglo XVIII (1702 – 1761) que elaboró lo que él denominaría Teorema de Bayes. Este científico formó parte de la Royal Society británica, entidad que reunía a las mentes inglesas de mayor prestigio en aras del avance científico para la sociedad.

El estudio de la probabilidad de Thomas Bayes posibilitó la creación del Teorema homónimo basado en el estudio de la probabilidad condicionada extendida a un amplio número de muestras. El teorema se enuncia de la siguiente manera:

Sea $\{A_1, A_2, \dots, A_i, \dots, A_n\}$ un conjunto de sucesos mutuamente excluyentes y exhaustivos, y tales que la probabilidad de cada uno de ellos es distinta de cero (0). Sea B un suceso cualquiera del que se conocen las probabilidades condicionales $P(B|A_i)$. Entonces, la probabilidad $P(A_i|B)$ viene dada por la expresión:

$$P(A_i|B) = \frac{P(B|A_i)\,P(A_i)}{P(B)}$$

Donde:

- $P(A_i)$ son las probabilidades a priori,
- $P(B|A_i)$ es la probabilidad de la hipótesis,
- $P(A_i|B)$ son las probabilidades a posteriori.

Esto significa que si tenemos la probabilidad anterior al suceso (paso anterior), la probabilidad de que se cumpla B si se da la situación del paso anterior y la probabilidad de que se cumpla B seremos capaces de estimar la probabilidad de que ocurra el paso siguiente.

Sin embargo, no toda la comunidad acepta que el Teorema de Bayes sea puramente probabilístico. Un sector de los profesionales de la probabilidad no acepta que las probabilidades se basen de alguna forma en lo que llaman probabilidades "subjetivas". Estos estadísticos tradicionales creen que el estudio debe basarse, por tanto, en probabilidades empíricas, demostradas experimentalmente.

De esta forma, una vez enunciado el Teorema de Bayes, nos falta un último paso y quizás de los más importantes, relacionar esta formulación a nuestro SLAM. Al fin y al cabo, nuestro problema SLAM significa que nuestro robot es incapaz de localizarse y desplazarse por un mapa.

Para poder afrontar este problema y solucionarlo nos debemos basar en nuestros actuadores y sensores, es decir, en los estímulos que reciba el robot. Sin embargo, esos estímulos no siempre son los esperados, entre otras razones, por el ruido de los mismos sensores. Existen varios motivos por los cuales un modelo nunca funcionará a la perfección en el mundo real, por ejemplo, en el caso del manejo de un robot, el procesador puede dar la orden de frenar las ruedas, sin embargo, por la velocidad que poseía o por imperfecciones del suelo éstas pueden patinar, y de ahí en adelante nuestro robot se encontraría mal situado.

Esta situación es la que nos obliga a hacer los sistemas en bucle cerrado en lugar de bucle abierto, para así podernos adaptar a la perfección a las circunstancias que nos rodean. Por ello, para estimar el error producido por todas esas casualidades que pueden ocurrir como también que la sensibilidad del sensor no sea lo suficientemente elevada y esto provoque problemas con nuestro conversor analógico-digital y, por tanto, fallos en las medidas.

Debemos ser capaces de cuantificar todos estos posibles errores, para los que usaremos las funciones de probabilidad, porque en el mundo no hay tanta perfección como para que se cumpla el modelo propuesto. Aquí es donde interviene la formulación de Bayes, para poder afrontar estos errores y ser capaces de resolver este problema.

De esta forma, si por ejemplo nuestra A del teorema de Bayes propuesto anteriormente fuera la posición de nuestro robot y la B fuera la información que tenemos de nuestro entorno, podríamos ir obteniendo más puntos en los cuales sabemos dónde hay o no obstáculos y así lograr un mapeo de una forma progresiva. Esto es la formulación de bayesiana para el SLAM.

Una vez obtenido esto, seremos capaces de modelar el sistema teniendo en cuenta nuestra incertidumbre gracias a este teorema probabilístico.

El siguiente paso que nos ocupará será el de plantear una forma de resolución de este problema, ya que todo lo anterior era para que pudiéramos manejar la incertidumbre provocada por el sesgo de medidas y actuadores.

Cabe decir que la formulación bayesiana no solamente aporta luz sobre el problema del SLAM, también es frecuente su uso en las implementaciones informáticas antispam, de forma que observa que es lo que uno suele archivar en el correo no deseado y el gestor de emails aprende secuencialmente qué debe meter en spam y qué no debe meter. Por tanto, aunque la comunidad científica no lo acepte al completo como un estimador "tradicional" de probabilidad la formulación bayesiana ha demostrado ser muy útil en varios ámbitos de la ciencia.

Nuestro siguiente paso que abordar será la resolución de nuestro sistema y para ello existen varias formas de resolución, todas ellas basadas en la probabilidad y que después de tantos años de aplicación han terminado dando muy buenos resultados y se han consolidado como los caballos ganadores a la hora de afrontar el problema SLAM. Estos son:

- **Filtro de Kalman Extendido**

El filtro de Kalman es un algoritmo probabilístico que desarrolló Rudolf Emil Kalman en el año 1960 orientado a poder medir las variables no observables de un sistema a partir de las observables del mismo contemplando los posibles errores de medición.

Para realizar este algoritmo se necesitarán dos tipos de ecuaciones, la primera de ellas deberá relacionar las variables de estado con nuestras variables observables, estas recibirán el nombre de variables principales; el segundo tipo de ecuaciones serán aquellas encargadas de relacionar las variables de estado con su evolución temporal, es decir, ecuaciones encargadas de evaluar las variables de estado en función del tiempo, éstas reciben el nombre de ecuaciones de estado.

El funcionamiento de este algoritmo se basa en una formulación recursiva en dos pasos. El primer paso obtiene el nombre de paso predicción ya que se basa en la estimación de las variables de estado en función de su propia dinámica. El segundo paso recibe el nombre de paso de corrección, donde corregiremos la predicción hecha en el paso anterior usando la información aportada por las variables observables.

El algoritmo procede de la siguiente manera:

1. Fase de predicción

Esta primera ecuación consiste en la estimación a priori del estado futuro:

$$\hat{x}_{k|k-1} = \Phi_k \, x_{k-1|k-1}$$

Siendo $\Phi_k$ la matriz de transición de estados encargada de relacionar el antiguo paso con el nuevo.

En esta segunda ecuación vamos a evaluar la covarianza del error de la estimación a priori:

$$P_{k|k-1} = \Phi_k P_{k-1|k-1} \Phi_k^T + Q_k$$

Siendo $Q_k$ la matriz de varianza del error de medición.

$P_{k-1|k-1}$ será por tanto la covarianza del error de la estimación a posteriori del paso anterior.

2. Fase de corrección

Esta segunda fase consistirá en 4 ecuaciones que nos otorgarán una estimación más acertada de las estimaciones hechas en el paso de predicción anterior

En esta primera ecuación vamos a calcular la actualización del residuo del error en la medida:

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1}$$

$H_k$ será la matriz que relaciona las mediciones y el vector del momento k, suponiendo condiciones ideales.

$z_k$ es el vector de mediciones en el momento k.

En esta segunda ecuación procedemos a calcular la ganancia K de Kalman:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$$

Donde $R_k$ es la matriz de la covarianza del ruido de las medidas

La tercera ecuación se corresponderá con el cálculo de la estimación a posteriori:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k$$

Y, por último, esta ecuación se encargará de calcular la covarianza del error asociada a la estimación a posteriori:

$$P_{k|k} = (I - K_k H_k) P_{k|k-1}$$

Después de varias iteraciones seremos capaces de obtener nuestra estimación de estados no observables en base a los observables y sus errores. El hecho de que esta formulación sea recursiva la hace muy atractiva para diversos experimentos, por ejemplo, para el estudio de econometría, donde ha demostrado ser muy útil.

Una de las mayores desventajas del Filtro de Kalman es que supone que todas las distribuciones de probabilidad que recibe las asume como Gaussianas, lo que puede provocar que en caso de que la distribución de entrada no sea la esperada haya fallos en el algoritmo y provoque un fallo del mismo.

También es necesario señalar para el asunto que nos atañe que esta formulación no resulta óptima en asuntos como los de navegación y localización ya que pueden existir sistemas que no se representen en la realidad como sistemas lineales, por lo tanto, para conseguir resolver este problema nuestro algoritmo se aproximará a través de series de Taylor lo cual nos permitirá contemplar también sistemas no lineales. A esto se le llamó Filtro de Kalman Extendido.

Este filtro de Kalman Extendido se basa en la suposición de que el sistema de entrada puede ser no-lineal. Esto provoca una pequeña alteración en los cálculos del mismo algoritmo, ya que algunas de nuestras ecuaciones deberán ser aproximadas por medio de series de Taylor. [11]

Estas alteraciones se diferencian con respecto las iniciales en que en nuestro paso de predicción deberemos realizar la linealización de términos lo que significa que realmente obtendremos aproximaciones de primer orden a los términos óptimos. Si el modelo no lineal difiere mucho del modelo gaussiano pueden producir falsas medias y covarianzas producto de esta aproximación, lo que se traduce en el mal funcionamiento del filtro, incluso la divergencia del sistema.

Otro punto para tener en cuenta es que al realizar nuestra aproximación de Taylor provocamos la aparición de matrices jacobianas que no son de sencilla resolución, de esta forma esto puede provocar un aumento en el tiempo de computación del sistema y por tanto dificultar su implementación en ciertos dispositivos y por otro lado el aumento de probabilidades de fallos matemáticos que son fáciles de cometer, pero difíciles de detectar.

A pesar de los inconvenientes reflejados anteriormente, es necesario decir que este método de recopilación de datos es el más ampliamente usado para solucionar el problema SLAM debido a su robustez a costa de los problemas que puedan surgir en la implementación del método.

- **Filtro de partículas:**

Este método se basa en la idea de obtener las variables de estado a través de obtener el peso (o relevancia) de cada una de las muestras tomadas de forma que logremos aproximar nuestra función de estado que no es de fácil obtención a otra que si conozcamos. Para obtener la sucesión de probabilidades este método se basa en el uso de los métodos de Monte Carlo, usados para hacer grandes cálculos numéricos y que mantienen las estadísticas que aportan los nodos. [12]

Para resolver este problema por tanto se propondrán diferentes distribuciones que nos ayudarán en la resolución de nuestro problema:

$\mathrm{p}(\mathrm{x})$ es la función a la que nos debemos aproximar ya que es una distribución de complicado muestreo.

$\pi(\mathrm{x})$ es una distribución evaluable en todos sus puntos.

$\mathrm{q}(\mathrm{x})$ es una distribución propuesta para la solución de nuestro problema.

$x^i \sim q(\mathrm{x})$ serán los puntos de muestra de la distribución propuesta.

De esta forma, la obtención de nuevos puntos para el estudio de la distribución objetivo se realizará a través de la siguiente ecuación:

$$p(x_{0:k} \mid z_{1:k}) \approx \sum_{i-1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i)$$

Siendo $x_{0:k}^i$ un conjunto de puntos aleatorios de la distribución propuesta.

Siendo $w_k^i$ el conjunto de pesos ponderados de cada punto.

Siendo $x_{0:k}$ el conjunto de estados de 0 a K.

Suponemos ahora que p(x) será proporcional a distribución π(x) y así podremos aproximar más aún la primera a la segunda. De esta manera seremos capaces de evaluar los pesos de cada una de las muestras Ns para construir nuestro sistema:

$$w^i \propto \frac{\pi(x^i)}{p(x^i)}$$

De esta forma podemos comprobar como los pesos estarán normalizados de forma que la suma de estos puntos será igual a la unidad:

$$\sum_i w_{k=1}^i$$

En el caso de que el caso de estudio sea secuencial, el caso que realmente nos importa, podremos calcular los pesos de nuestros puntos en función de su relevancia en el paso anterior del algoritmo con la siguiente función:

$$w_k^i \propto w_{k-1}^i \frac{p(z_k \mid x_k^i) p\left(x_k^i \mid x_{k-1}^i\right)}{q\left(x_k^i \mid x_{k-1}^i, z_k\right)}$$

De esta forma el procedimiento iterativo se basará en primera instancia en buscar nuestros puntos de muestra en la distribución propuesta para, posteriormente, compararlo con nuestra función evaluable de forma que nos devuelva su peso en la distribución. El proceso acabará una vez que se evalúen todos los puntos previstos (Ns).

Puede ocurrir que, cuando se realicen muchas medidas, llegue un momento en el que todos los puntos alcancen pesos despreciables excepto uno, a este suceso se le llama degeneración de la partícula.

Existen distintas formas de evitar que ocurra este suceso que nos conducirá a fallos en las medidas, sin embargo, la experiencia nos ha demostrado que la más útil es el remuestreo de las partículas.

El remuestreo se basa en volver a recorrer todos los puntos que hemos obtenido anteriormente para dar prioridad a las partículas que tengan un mayor peso y así ser capaces de afinar la aproximación y evitar problemas de degeneración de partícula.

Un buen marcador para controlar esta degeneración y que nos servirá para el remuestreo será la variable $N_{eff}$, que nos mostrará la eficiencia que está teniendo nuestro proceso de forma que funcione como una bandera para poder terminar nuestro remuestreo. La variable se obtendrá de la siguiente forma:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2}$$

Por tanto, podemos ver que un $N_{eff}$ bajo significa que esos pesos han perdido consistencia y esto provoca un empobrecimiento de las medidas.

El método a seguir en este caso comenzará igual que antes de que existiera remuestreo, se van escogiendo puntos de la distribución a testear en bucle hasta que logramos el número de muestras deseado. A partir de este instante el funcionamiento del algoritmo se extiende. Obtendremos el peso total de nuestras partículas y posteriormente las normalizaremos con respecto a ese peso total, finalmente comprobamos si $N_{eff}$ es mayor que nuestro umbral. En caso afirmativo saldremos de nuestro bucle y daremos por terminado el funcionamiento del filtro, en caso contrario, será necesario un remuestreo, que se basará en repetir lo anterior, pero con una diferencia, se sustituirán las partículas con menor peso normalizado por las que lo tengan mayor, dando así consistencia a los datos obtenidos.

Aunque este método no es el más popular en la comunidad tiene una amplia aceptación y uso, aunque suele surgir un problema cuando este filtro se encuentra en superficies de alta dimensionalidad.

De esta forma el teorema de Rao-Blackwell nos ayuda a superar este problema proponiendo que lo que se debe hacer es dividir el mapa en dos subespacios, uno que se actualice analíticamente y otro que se actualice a través del filtro de partículas propuesto anteriormente.

Por otro lado, este método también tiene ventajas como el hecho de no ser paramétrico que no les exige a los modelos que deban ser lineales para su resolución, un bajo coste computacional y que es un método adaptativo, contra más partículas mejores resultados obtendremos ya que aumentamos la densidad probabilista.

- **Grafos:**

Mientras que la solución a los problemas SLAM basadas en filtros están preparadas para la resolución del Online SLAM, las soluciones gráficas o Graph-SLAM se usan para resolver los problemas de Full SLAM, como hemos comentado anteriormente.

Como se ha comentado anteriormente sobre los mapas topológicos, el Graph-SLAM se basará en el designio de las landmarks, cuya posición se asociará con los nodos y las órdenes de su tarea a cometer serán codificadas en las aristas del mismo grafo.

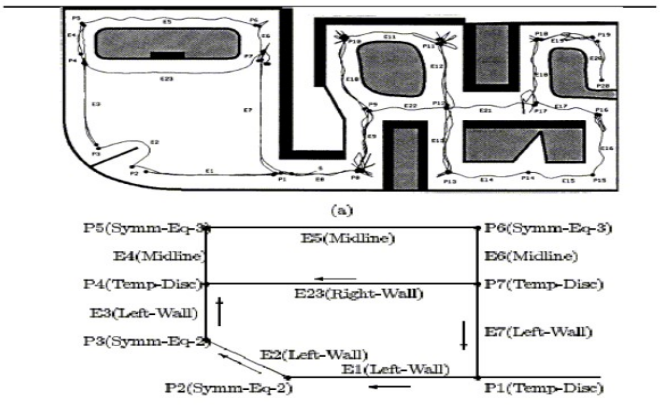



Figura 7. Encima el mapa real recorrido. Debajo el mapa topológico deducido.

Un detalle importante a tener en cuenta es que este tipo de SLAM son lo que se llama un “lazy algorithm”, es decir que pospone los cálculos y no los realiza todos a la vez. Por ello, la forma de abordar esto se dividirá en dos módulos, el front-end y el back-end.

El módulo front-end se encargará de crear el grafo gracias a las restricciones que es capaz de percibir a través de los sensores y la odometría que se le facilita al robot, la información que usa él mismo para estimar su posición, tal como el giro de las ruedas o su diámetro. Por otro lado, el módulo back-end, por tanto, tendrá como tarea optimizar el recorrido o simplemente la creación de este grafo. Al compararlo con el típico Online SLAM, se puede apreciar que el cálculo se torna bastante más costoso, ya que al ser un problema de Full SLAM debe contemplar una mayor cantidad de datos.

Por último, es necesario mencionar la forma de construir mapas basados en el principio de la rejilla de ocupación. Este método destaca porque consigue dividir la superficie en cuadrados de igual tamaño e irá ocupando las celdillas correspondientes con 0 y 1 dependiendo de si están ocupadas o no. Sin embargo, fuera de la teoría nuestro robot no puede estar 100% seguro de lo que percibe debido a fallos de sensado y odometría, por ello, en lugar de en cada celdilla colocar un 0 o un 1, pasaremos a introducir en cada una de estas celdillas una función de probabilidad que nos puede otorgar cierta luz sobre si esa celdilla es un obstáculo o no. Además, el correcto funcionamiento de esta metodología de resolución está directamente ligado al tamaño de las celdillas. Si buscamos un alto nivel de detalle y exactitud sobre nuestro mapeo necesitaremos reducir el tamaño de esas celdillas al mínimo, pero eso aumentará a un nivel superlativo el coste de computación dificultando en gran medida la resolución de nuestro mapeo.

La odometría es el estudio de la estimación de la posición en la que se encuentran un sistema que se desplaza. Por tanto, los datos odométricos será toda la información que el robot necesita para poder estimar correctamente su posición y actuar en consecuencia.

Con respecto a la odometría existen dos tipos de errores asociados:

- Los errores sistemáticos: estos errores se caracterizan por pertenecer al sistema, es decir, están asociados a nuestro sistema y el no tenerlos en cuenta o cometerlos puede resultar sumamente graves para la obtención de buenos resultados. Algunos ejemplos pueden ser la discordancia entre diámetros de ruedas o su mal alineamiento.

- Los errores no sistemáticos: por otro lado, este tipo de errores están asociados con situaciones relacionadas con el entorno y que pueden ser complicadas de evitar como el derrape de una rueda o el deslizamiento sobre una superficie demasiado deslizante.

El buen estudio de la odometría proporcionará mucha más robustez a cualquier algoritmo, pues facilitará la tarea de localización de nuestro sistema en el entorno que le rodea.

### a. Método del camino más corto

Anteriormente hemos repasado el estado actual en el que se encuentran el mapeo y la localización para robots autónomos. En este punto cambiamos nuestra dirección a otro tipo de algoritmos, los del Camino Más Corto.

Estos métodos no son más que problemas de optimización aplicados a grafos, en los cuales lo que queremos es llegar de un punto a otro en el menor tiempo de cómputo, teniendo

en cuenta el peso de cada uno de los nodos. Un grafo no es más que un conjunto de nodos y aristas, la utilidad práctica de estos elementos consiste en la representación gráfica de sistemas interconectados entre sí mismos y comprobar así, como se interrelacionan los pares de módulos. Sin embargo, no todos los grafos son iguales, existen varios tipos, pero solamente hablaremos de dos de ellos:

- Grafos dirigidos: en este tipo de grafos la conexión entre nodos está realizada por medio de "vectores", es decir, las aristas tienen una dirección que debe ser seguida obligatoriamente. Es decir, en la figura 3 los nodos 6 y 4 sólo podrán unirse en una dirección predefinida o de 6 a 4 o de 4 a 6, pero no al contrario del marcado con anterioridad.

- Grafos no dirigidos: por el contrario, en esta clase de grafos la arista quedará definida por dos puntos sin orientación exigida. En este caso si el grafo de la figura 3 fuera no dirigido la conexión entre, por ejemplo, 6 y 4 no importaría el sentido que tuviera la conexión.

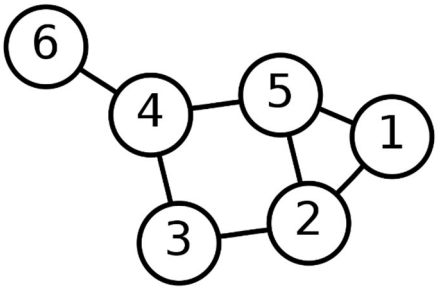



Figura 8. Grafo

A lo largo del tiempo han ido surgiendo distintos métodos de resolución para este problema que se nos plantea. Estos son los más conocidos:

❖ Algoritmo de Dijkstra

En este algoritmo creado por Edsger Dijkstra en 1959 recorremos todos los nodos del grafo y los iremos marcando a medida que los vayamos investigando. Esta investigación se basa en situarse en un nodo, comprobar las distancias de los nodos vecinos al marcado comenzando en el origen y comprobar si esta distancia es menor que la guardada anteriormente en memoria, en caso de ser menor se sustituirá y pasaremos al siguiente nodo de forma recurrente hasta llegar a completar todo nuestro grafo. A la distancia existente entre el origen a los nodos colindantes al marcado se le denominó distancia tentativa. Este método es el más extendido en la comunidad, sin embargo, no es útil para grafos en los que existen nodos aislados, como el de la Figura 8.

Para explicar correctamente este algoritmo lo haremos con un ejemplo.

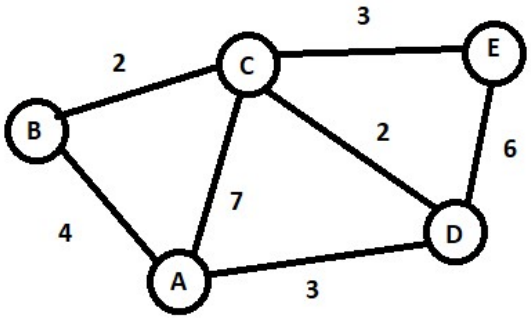



Figura 9.

A la vista de este grafo, nosotros comenzaremos por el nodo que veamos oportuno, en este ejemplo comenzaremos con el A. En primer lugar, lo que deberemos asignarles a todos los nodos es un valor de infinito, para cuando así actualicemos por su verdadero valor siempre podamos actualizarlo. El nodo A se quedará con valor 0 ya que empezamos por él.

Nuestro siguiente paso será comprobar cualquiera de los nodos que A tiene alrededor (B, C o D). Por ejemplo, escogemos el B. Para comprobar su peso lo que haremos será sumar la distancia tentativa de A, es decir la distancia que existe del nodo actual al nodo inicial, a el peso de la arista, que en este caso será 4, por tanto, el nuevo peso del nodo B será 4.

Una vez hecho esto pasamos al siguiente nodo colindante a A, por ejemplo, C. En este caso C tomará el valor de 7, y seguidamente D tomará el valor de 3.

Ahora que ya hemos comprobado todas las distancias que rodean a A podremos marcar a este nodo como visitado y pasaremos al siguiente nodo con menor distancia relativa y que no esté visitado. De esta forma, pasaremos a trabajar con el nodo D que tiene una distancia al origen de 3, la menor.

Por tanto, volvemos a comenzar, revisar los nodos no visitados colindantes, es decir, C y E. Cogeremos en primer lugar el nodo C, y para obtener su distancia desde el origen a este punto sumaremos como antes la distancia tentativa al peso de la arista, la diferencia ahora está en que esta distancia tentativa no es nula, si no que vale lo que hay del origen a el nodo actual, 3, por tanto, la suma de ese peso será 3+2=5. Puesto que este nuevo peso es menor que el que tenía C anteriormente (7) se sustituirá el anterior por el nuevo de forma que su peso pasará a ser 5. Repetimos procedimiento ahora con el nodo E, y obtendremos que su primer peso será 3+6=9. Ahora el nodo D está visitado, pues todos éste ha visitado a todos sus colindantes.

En este momento, abordaremos el nodo B, el siguiente con menor distancia desde A, el único colindante no visitado a B es C, así que procedemos a calcular la distancia de A a C pasando por B, y ésta resulta 4+2=6. Casualmente, este peso es mayor que el que ya tenía anteriormente el nodo C, por tanto, no se actualizará y marcaremos B como visitado y pasaremos a comprobar C.

El único nodo colindante no visitado a C será E con su peso de 9. Por tanto, sumaremos de nuevo la distancia tentativa de A a C (5) al peso de la arista que une C y E (3), que no dará que la distancia de 8 es menor a la distancia que ya poseía B, de esta forma, se actualiza el nodo E, y puesto que es el único no visitado vamos a él pero cuando llegamos a él comprobamos que todos los de alrededor están visitados y terminamos ejecución.

Las distancias de todos los nodos a A han quedado de la siguiente manera:

- A - B: 4
- A - C: 3
- A - D: 5
- A - E: 8

Como se puede comprobar este algoritmo lo único que hará será buscar el camino óptimo desde un nodo inicial a todos los demás sin darle relevancia a las distancias que puedan existir entre los nodos que no sean el camino más corto.

Debido a su simplicidad es, con diferencia, el más usado en el aspecto de navegación y localización, sin embargo, esto restringe su uso ya que no contempla el hecho de que alguna arista tenga peso negativo, cosas que no puede ocurrir físicamente hablando de distancias, pero sí en disciplinas de la economía. También es un punto en contra el hecho de que no pueda funcionar en grafos que tengan algún nodo aislado, ya que hemos visto que la condición de final de nuestro algoritmo será que todos los nodos de alrededor estén visitados, y en caso de nodo aislado, puede terminar la ejecución antes de lo esperado.

❖ Algoritmo de búsqueda A*

Este algoritmo es bastante distinto a los demás, puesto que no se basa sólo en la heurística del experimento, sino que también tiene en cuenta el coste real que costará el recorrido. El desarrollo de este algoritmo se basa en que contempla dos funciones de costes, la primera será el coste del recorrido del nodo actual hasta el final (heurística), la segunda contemplará el coste de haber llegado a esa posición. A través de dos estructuras de datos auxiliares ordenará en una los nodos ya comprobados y en la otra los datos por comprobar.

❖ Algoritmo de Floyd-Warshall

Este método tiene un mayor coste computacional, sin embargo, nos ofrece la distancia menor que existe en un grafo entre todos los pares de nodos de nuestro grafo. Su funcionamiento se basa en ir comprobando cuál es la menor distancia existente entre cada par de nodos recorriendo todos y cada uno de los caminos posibles de forma recursiva, una vez que terminar con un par de nodos pasa al siguiente par y así sucesivamente.

Existen más algoritmos para resolver el problema del camino más corto en grafos, sin embargo, para nuestro estudio no son relevantes puesto que contemplan pesos negativos en las aristas, situación que no podemos contemplar ya que esas serían las distancias por recorrer por nuestro robot autónomo. Por ejemplo, el algoritmo de Bellman-Ford es uno de los más usados puesto que se desarrolla igual que el método de Dijkstra, pero teniendo en cuenta la posibilidad de que se produzcan ciclos negativos; el algoritmo de Johnson no es más que una mezcla de los algoritmos de Bellman-Ford y Dijkstra para así optimizar el de Floyd-Warshall; y, por último, el de Viterbi que usa métodos probabilísticos para la resolución de nuestro problema.

# 3. Descripción del Hardware

Ya hemos mostrado en qué punto se encuentra el arte de la programación y navegación robótica, en este apartado del Trabajo de Fin de Grado describiremos brevemente el hardware sobre el que implementaremos nuestro software, incluyendo el microcontrolador, las ruedas, los sensores y la fuente de alimentación que usaremos para el correcto funcionamiento de nuestro coche.

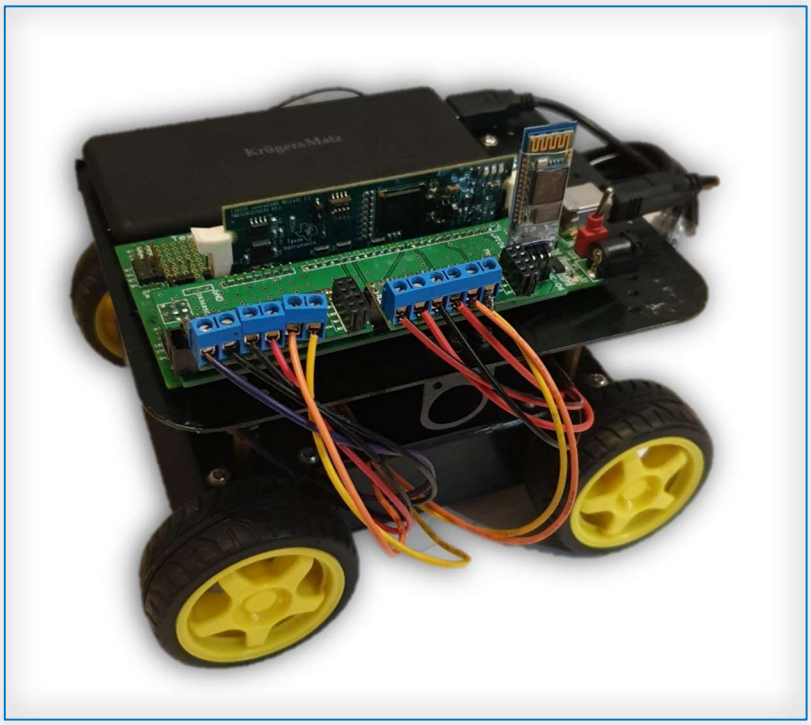


Figura 10. Imagen del coche con el que trabajaremos.

Como podemos ver en la imagen, en la parte superior de nuestro robot se sitúa nuestro controlador, el TMS320F28335 de Texas Instruments Inc., que será el que nos ayude a manejar el recorrido de nuestro coche.

El TMS320F28335 es un DSP (*digital signal processor*, en español, procesador de señal digital) con un uso muy amplio en el mercado gracias a diversos factores, como, por ejemplo, el entorno de desarrollo que nos facilita Texas Instruments para implementar nuestros protocolos informáticos, el Code Composer Studio para C/C++.

A parte del fantástico IDE que nos da esta empresa, el microchip tiene unas características muy notables como 256k x 16 de memoria FLASH, el mayor almacenamiento de los procesadores de esta gama. Por otro lado, la memoria volátil será de tipo SARAM y será de 34k x 16. Estos dos espacios de memoria darán al controlador una gran versatilidad en el sector de la computación.

El conversor Analógico-Digital con sus 16 canales de 12 bits nos ayudará en gran medida a que nuestro controlador sea capaz de leer medidas captadas por sus sensores con gran precisión, si unimos este detalle con el hecho de que también contamos con 64 pines de entrada

y salida podremos afirmar que nuestro controlador será capaz de mantener una relación idónea con el entorno. [15]

Por otro lado, cabe señalar también el hecho de que este controlador posee 6 accesos directos a memoria (DMA), para facilitar la interacción de ésta tanto con el ADC, como con, el XINTF, el McBSP o la SARAM. Este procesador además tendrá 8k x 16 Boot ROMs para encender los distintos sistemas periféricos a través del puerto serie, que son:

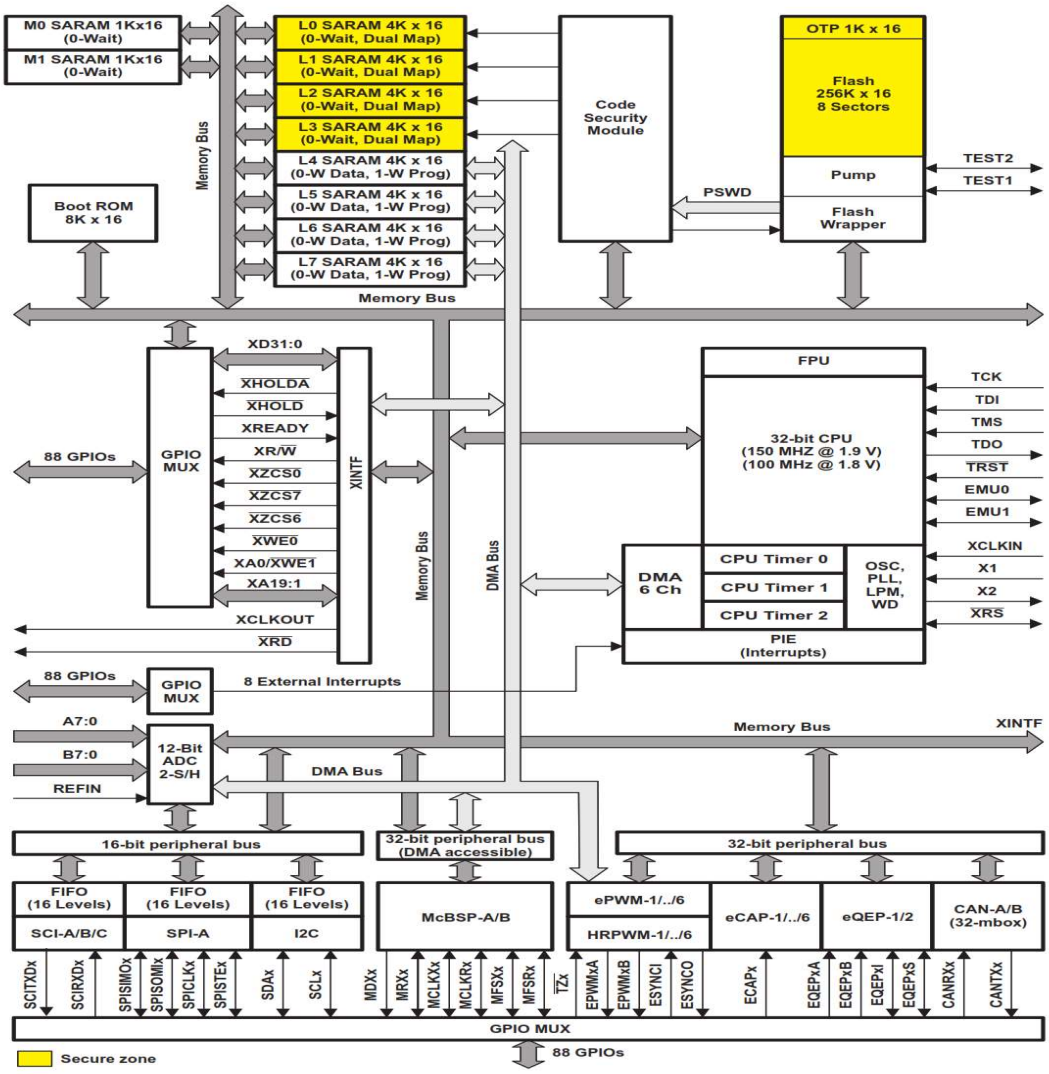



Figura 11. Diagrama funcional de bloques del controlador.

- Hasta 2 módulos CAN.
- Hasta 3 módulos SCI (a través de la UART).
- Hasta 2 módulos McBSP (Configurables como SPI).
- 1 módulo SPI.

- 1 bus I2C.

En la página anterior se puede apreciar un diagrama de bloques que nos enseña cómo se interconectan los distintos periféricos con la CPU y las distintas memorias.

Este DSP se conectará a la placa Experimenter's Kit USB Docking Station de la familia C2000 de Texas Instruments cuyo uso es facilitar la comprensión y desarrollo de cualquier código informático que se desee implementar en un sistema futuro. Tiene grandes aplicaciones educativas gracias a sus periféricos como leds que nos indican el estado entre otras cosas. También nos facilita el hecho de dar tensión a nuestro controlador para ponerlo en funcionamiento, gracias a una entrada que éste posee y nos permitirá dar una tensión al dispositivo de 5V o 3,3V.

Para el control de los motores se ha usado el módulo DVR8835 de Texas Instruments, el cual se basa en el funcionamiento de un puente H para controlar las ruedas. Esta placa será capaz de controlar 2 ruedas simultáneas, por tanto, será necesario conectarlo a dos salidas distintas para controlar las 4. Por otro lado, se configurarán 4 PWMs para controlar cada una de estas ruedas y otros 4 pines de E/S para darles la dirección a éstas. Debido al control de estas ruedas la demanda energética de nuestro coche aumenta, de tal forma, la tensión que será necesario darle será de 7.5V. [14]

Por otro lado, cuando comenzamos a armar el coche nos dimos cuenta de que el hecho de cablear todo nuestro coche se tornaba en una situación compleja y que esto provocaría ruido en las órdenes y señales de nuestro sistema. Por tanto, se decidió crear una placa PCB para facilitar todas nuestras conexiones y así evitar los problemas planteados anteriormente.

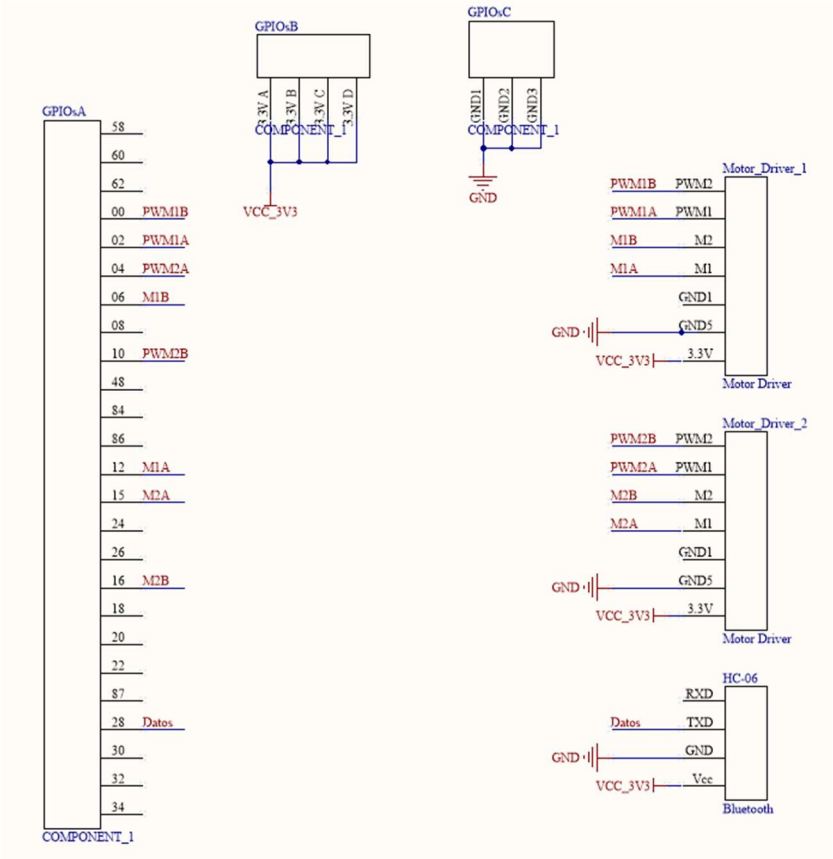



Figura 12. Esquemático de nuestra PCB.

Una vez definidas las conexiones que se debían realizar, comenzamos a desarrollar la placa a través de Eagle y este es el resultado final:

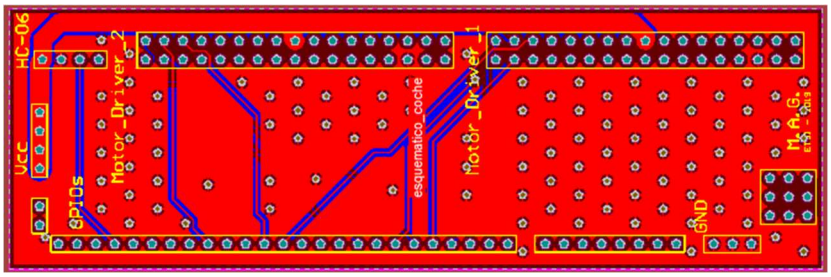



Figura 13. Diseño final de la PCB.

Gracias al diseño e introducción de la PCB en nuestro sistema conseguimos obtener mejores resultados de transmisión de información y señales en nuestro sistema, en resumen, fue un gran avance.

Es necesario hablar también de los motores que impulsan nuestras ruedas, [16] las cuales poseen un diámetro de 6.5 cm. Estos motores serán los TT Geared Motor with Back Shaft que, como hemos dicho anteriormente tendrán un rango de tensiones de unos 7.5V, por ello, deberemos alimentarlos a esa tensión. Además, debemos añadir que estas ruedas solo se mueven en una dirección, pero en ambos sentidos, es decir, hacia adelante y hacia atrás, por tanto, para hacer que nuestro vehículo gire lo que nosotros deberemos hacer es manejar las cuatro ruedas para conseguir ese giro.

Por ejemplo, si nosotros queremos girar a la izquierda deberemos decirle a nuestro coche que active las dos ruedas de la derecha hacia atrás y las dos de la izquierda hacia adelante, de forma que se consigue así un giro de 90º.

Un punto muy relevante a la hora de convertir el sistema en autónomo es el de los sensores, que, básicamente, constituirán lo que a los seres humanos serían los cinco sentidos.

Para este TFG se ha decidido implementar un sensor de proximidad colocado delante del robot que nos indique los posibles obstáculos que se pueda encontrar el sistema.

- ***Sensor de proximidad: HC-SR04***

Este sensor será usado para detectar obstáculos y almacenarlos en el mapa que, posteriormente, recorrerá el algoritmo principal del proyecto. Una vez que el sensor registre un valor de distancia menor que el umbral se dará la señal de obstáculo y se procederá a sortearlo.

Se ha decidido escoger el HC-SR04, que ha demostrado ser un sensor de proximidad muy fiable. Como se podrá apreciar en la siguiente imagen este sensor consta de un emisor y un receptor.


Figura 16. Sensor HC-SR04.

El funcionamiento del mismo es simple, el emisor enviará una señal de ultrasonidos de forma que si hay un obstáculo cercano ésta rebota y la recibe el receptor. Una vez hecho esto, ese tiempo de diferencia entre emisión y recepción será introducido en la siguiente fórmula:

$$Distancia = \frac{\Delta T_{ECO} * V_{sonido}}{2}$$

Este dispositivo estará alimentado a 5V a través de su pin VCC y a tierra a través del GND. Por otro lado, se conectarán los pines Trig y Echo a dos GPIO de propósito general como GPIO19 y GPIO27, respectivamente. El primero de los dos anteriores lanzará una señal que será la que active la verificación por parte del sensor de la distancia a la que se halla éste del obstáculo, el cometido del segundo Pin será tomar el dato que devuelve nuestro sensor.

Lo último a destacar de este sensor es que posee un rango de percepción de distancias de 2cm a 4m con un error de 3mm, lo que lo convierte en un sensor adecuado para la tarea que se debe realizar.

Ahora que ya ha quedado descrito nuestro robot en el siguiente punto introduciremos la nueva idea de elección y recorrido del camino más corto, para posteriormente, comentar los resultados y las conclusiones que hemos sacado de este Trabajo de Fin de Grado.

# 4. Implementación y desarrollo del Software

En base de todo lo que hemos presentado con anterioridad en este Trabajo de Fin de Grado proponemos un método alternativo para conseguir encontrar el camino más corto que atraviesa una superficie.

Nuestro método se basa, en parte, en el algoritmo de Dijkstra, según el cual, dada una matriz de 0 y 1 se intentará buscar el camino que recorra menos casillas para llegar a la última fila de ésta. En este caso, nuestra idea se basa en que las casillas que están ocupadas por un 0 están libres de obstáculos, de forma que las que tengan un 1 conformarán un obstáculo en nuestro mapa.

Este algoritmo es, conceptualmente, bastante sencillo, a medida que avanza a través de la matriz se va preguntando si delante, a su izquierda o a su derecha tiene un 0, es decir, camino despejado. De esta forma irá avanzando a través de la matriz hasta que llegue a una posición cualquiera de la última fila de la matriz. Así, se dará por concluido el trabajo de encontrar el camino más corto, posteriormente, a este robot se le dará la orden de recorrer este camino más corto a través de una serie de funciones que explicaremos con posterioridad.

Puede ser que, en una superficie amplia y de gran tamaño, no exista un solo camino, sino varios. Esto lo contempla nuestro código ya que se basa en dos estructuras iterativas anidadas a través de las cuales comprueba todos los caminos posibles y el mismo recorrido por dentro de nuestra matriz objetivo.

Antes de analizar este código en profundidad introduciremos una serie de opciones y posibles usos para implementar el mismo, para así demostrar su utilidad y versatilidad en nuestro mundo:

- En zonas catastróficas donde el acceso humano es imposible debido a la existencia de radiación y es necesario hacer un reconocimiento o entregar algún tipo de medicamento a personas encerradas.

- Abastecimiento de pacientes por parte del personal médico en caso de que éste sea portador de una enfermedad altamente contagiosa, como el caso del ébola o el coronavirus que actualmente está provocando tantos problemas en nuestra sociedad.

- Guiado para sistemas de mantenimiento y reparación de sistemas de ventilación o de gas, debido a su estructura casi laberíntica y la dificultad que existe para encontrar obstáculos.

- Conducción de transportes encargados de realizar obras en sitios públicos, como, por ejemplo, pintar las líneas continuas o discontinuas de una carretera de una forma autónoma.

- También se podrá realizar un mapeo previo de tierras de cultivo y enviar a las distintas máquinas a diferentes zonas del terreno para realizar las funciones necesarias para mantenimiento o sembrado.

También cabe destacar, antes del análisis del código, las ventajas y desventajas que puede tener el hecho de implementarlo en un robot para que éste funcione de forma autónoma y así optimizar sus funciones:

❖ Ventajas

- Es un código poco pesado, lo que permite su implementación sin tener que preocuparnos en exceso por el espacio que ocupe en el almacenamiento del robot con el que se combine, lo que le dotará, en consecuencia, de alta compatibilidad.

- Debido a su modularidad sería sencillo extraer parte de este código para así combinarlo con otros programas distintos.

- Funciona para matrices de todos los tamaños gracias a la forma de asignación de las variables en el código.

❖ Desventajas

- No contempla el 100% de caminos posibles, puesto que no mira hacia atrás y se pueden escapar algunos caminos, aunque es poco probable.

- Debe existir un procesado previo de los datos del mapa por medio de cualquier herramienta que nos exporte la matriz necesaria para realizar el código, si no se realiza un mapeo previo de la superficie.

Una vez que ya se han expuesto los pros y contras de este código y sus posibles aplicaciones comenzamos a desarrollar en qué consiste el código.

Como hemos dicho anteriormente este código se compone de dos estructuras iterativas anidadas que se encargarán de recorrer la matriz buscando un camino cualquiera y de comprobar si quedan o no más caminos posibles por recorrer.

El proceso, concretamente, comenzará entrando en un bucle do-while, que servirá para comprobar los caminos por recorrer restantes, comprobando que la variable caminos es mayor que 0. Posteriormente, entraremos en la segunda estructura iterativa que será un while, cuya condición de reentrada será o que hayamos llegado a la parte más alta de la matriz, o que en esa vuelta en concreto no haya camino, es decir, aux=0.

Dentro de este segundo bucle while se procederá de la siguiente forma:

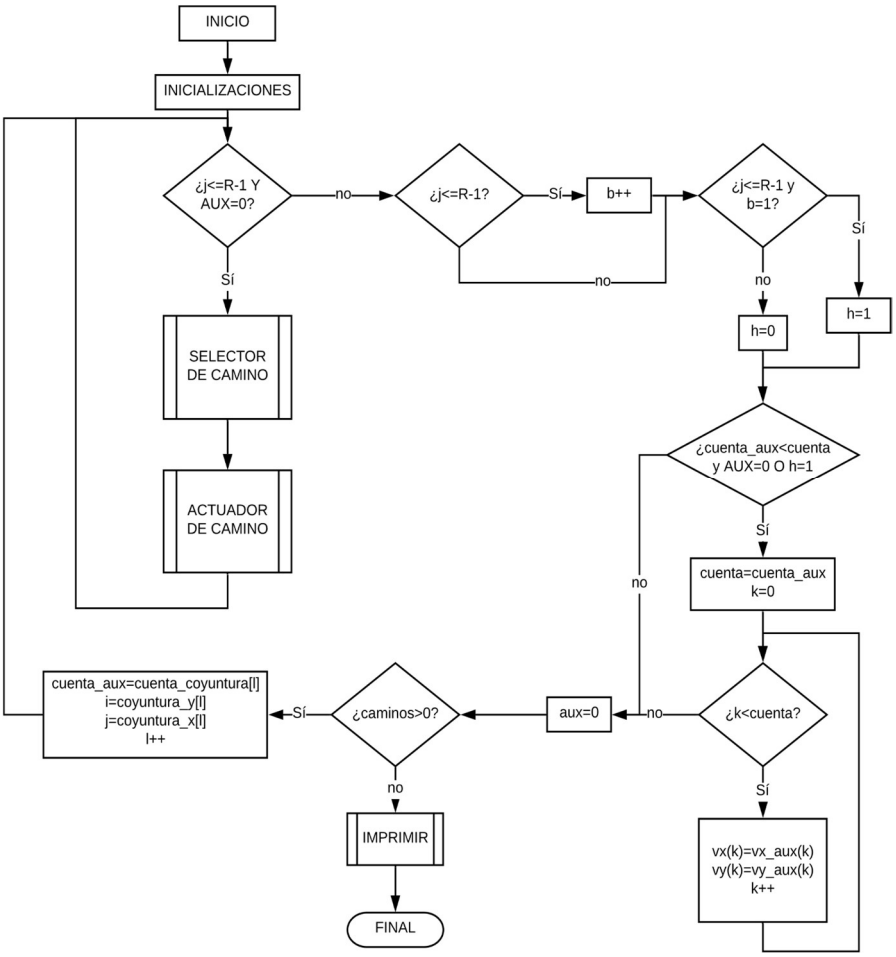
1. Se comprobará cual es la situación a abordar en el siguiente paso a través del módulo SELECTOR DE CAMINO, asignándosele a una variable un valor u otro dependiendo de cuál sea el próximo paso a dar, del 0 al 7.

2. El siguiente paso será acceder al módulo ACTUADOR DE CAMINO donde daremos las instrucciones de qué se debe hacer para seguir construyendo el mapa.

3. Gracias al módulo ACTUADOR DE CAMINO habremos actualizado las variables de posición y procederemos al inicio del bucle while de nuevo, de forma que se complete la estructura iterativa

Una vez hayamos llegado al final de la matriz, la última fila, podremos salir del bucle. En este momento se establecerán una serie de estructuras condicionales para comprobar si es la primera vez que se ha recorrido con éxito la matriz para así actualizar el valor cuenta, que será la variable que me marque qué camino es más corto. Por tanto, cada vez que acabo mi recorrido aumento la variable b que me dirá cuántos caminos completos he conseguido, y después

preguntaremos si es la primera vez para actualizar la variable *h* que será la que indique si es la primera vez que completo un camino o no.




Conocido el dato de si nos encontramos en la primera vuelta o no, se procedería al paso de actualización de las variables, es decir, en caso de que sea la primera vuelta o de que la variable cuenta, que como hemos dicho es la que marca cuál es el camino más corto, sea menor de la que estaba anteriormente entraremos en una estructura que copiará los datos de las variables auxiliares a las definitivas, para así tenerlas almacenadas en caso de que no hubiese más caminos o terminará la ejecución. Cabe añadir que para entrar en esta condición también es necesario que la variable *aux* esté a 0, que nos indicará que el camino anterior se ha terminado y, por tanto, es un camino completo. Podría ocurrir que uno de los caminos estuviera bloqueado y puesto que, seguramente, tendría el valor más bajo de cuenta haría que el código se actualizase cuando no debe, por esto se incluye esta variable en la pregunta.

Una vez que hemos actualizado o no nuestras variables ponemos la variable *aux* a 0, de forma que al volver a entrar no salga abruptamente del código y se produzca un bucle infinito. Posteriormente, comprobaremos si quedan más caminos. En caso afirmativo se terminará la ejecución del *do-while* y una vez impresos los resultados se terminará el programa. En caso negativo se comprobará el siguiente punto de coyuntura.

Se han denominado puntos de coyuntura a todas las bifurcaciones o trifurcaciones que puedan darse en nuestro mapa. Los puntos de coyuntura suponen un apartado clave en nuestra

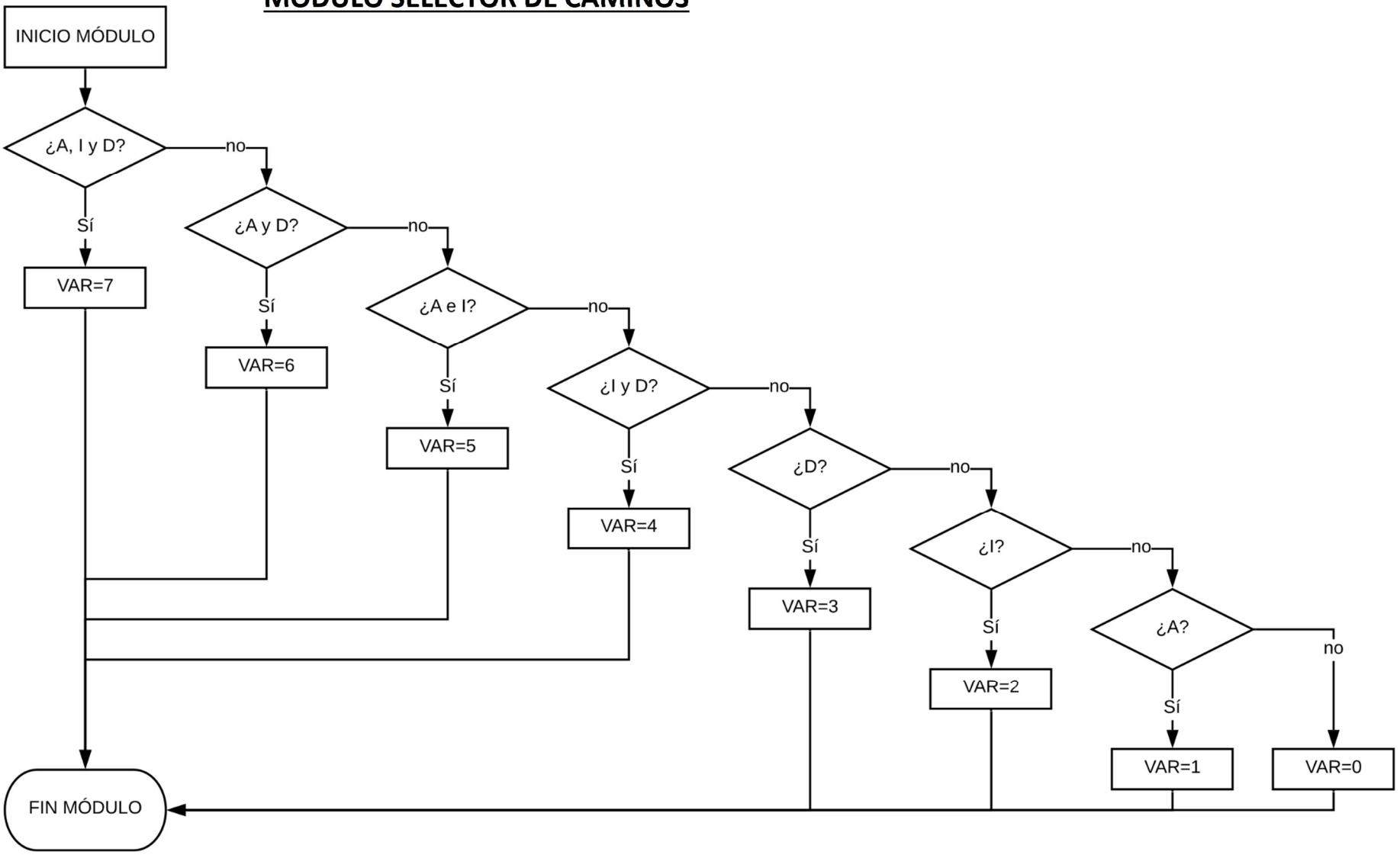
ejecución, ya que gracias a ellos nosotros conseguiremos almacenarlos y así tener referencias de dónde se hallan y en cuántos caminos divergen.

De esta forma el siguiente paso a dar si resulta que quedan más caminos por recorrer es el de actualizar el siguiente punto de coyuntura, es decir, habiendo almacenado todos los pasos anteriores a este punto, volveremos a comenzar desde el que se ha producido la bifurcación del camino, de forma que se reduce el tiempo de ejecución y nos saltamos casillas ya recorridas, que no aportarían nueva información.

Por tanto, volveremos a comenzar la sentencia while hasta completar el camino o descubrir que ese no sea uno posible, para después comprobar si ese es el más corto y actualizarlo o no, y así hasta que concluyamos que no quedan más camino y así tengamos nuestro camino más corto.

Ahora que el código ha sido comentado en su superficie nos adentraremos en él por medio del análisis de los dos módulos presentados en el diagrama de flujo anterior, el SELECTOR DE CAMINOS y el ACTUADOR DE CAMINOS.

## MÓDULO SELECTOR DE CAMINOS



En este módulo, como hemos comentado anteriormente, nuestro trabajo será comprobar cuál es el siguiente paso que debemos realizar, si debemos girar a la izquierda, derecha o seguir hacia adelante. Las opciones son las siguientes:

0. No hay camino.
1. Sólo hacia adelante.
2. Sólo hacia la izquierda.
3. Sólo hacia la derecha.
4. A la izquierda y a la derecha.
5. Hacia adelante y a la izquierda.
6. Hacia adelante y a la derecha.
7. Hacia adelante, a la izquierda y a la derecha (trifurcación).

La ejecución comenzará, como se puede ver en el diagrama de flujo anterior, comprobando si el siguiente paso es el que más bifurcaciones tiene, es decir el 7, e irá buscando posteriormente caminos con menos posibilidades a medida que avance. De esta forma, el algoritmo tendrá una forma de cascada, nos irá preguntando si el siguiente paso cumple la condición, en caso afirmativo, a la variable VAR se le asigna el número correspondiente; en caso negativo, se pasará al siguiente paso, y así sucesivamente, hasta que en caso de que se llegue al final y no se haya cumplido ninguna posibilidad se le asigna un 0 que nos dirá que no hay camino posible.

Esta estructura se ha escogido de esta forma debido a que en caso de que, por ejemplo, se pueda ir a la izquierda y hacia adelante, si estuviera estructurado al revés eso provocaría que entrara precipitadamente en un camino sin ser realmente el camino que debe tener en cuenta. Esta estructura consistirá en un if-else anidado para ir excluyendo distintas posibilidades y asegurarnos de que no nos metemos en posibilidades no adecuadas.

La forma de realizar estas comprobaciones es la siguiente:

- Para ir hacia adelante pregunto si M[j+1][i] es igual a 0 de forma que así sabré si la posición que tengo en frente está ocupada o no.
- Para ir hacia la izquierda pregunto si M[j][i+1] es igual a 0 y así sabré si puedo girar a la izquierda.
- Para ir hacia la derecha pregunto si M[j][i-1] es igual a 0 de forma que así sabré si la posición de la derecha está ocupada o no.

Las variables i y j serán las encargadas de ir actualizándose y guardando su valor instantáneo en mis variables vx_aux y vy_aux, éstas me servirán por tanto como señalizadores de dónde se hallará mi ejecución. La j será la que me muestre en qué fila se encuentra mi robot y la i la que me muestre la columna respectiva.

Pretendiendo una correcta ejecución en las preguntas para saber cuál es el siguiente paso que dar en éstas se incluyen también una serie de variables que nos indiquen que estamos dentro de los extremos de la matriz y también nos avisen de cuál ha sido el paso anterior, ya que es algo que nos interesa cuando manejamos las distintas posibilidades que nos puede ofrecer un camino.

Por ejemplo, si nuestro robot avanza en dirección lateral, es decir, en una misma fila, moviéndose, por ejemplo, de izquierda a derecha y no existen bifurcaciones se producirá una alteración en la decisión de camino a tomar pues cuando avance hacia la derecha nuestro robot se encontrará con un camino libre a la izquierda y otro a la derecha, por tanto creerá que se halla en el paso 4 (bifurcación a izquierda y derecha) en lugar de en el paso 3 (movimiento sólo a la derecha). Esto provocaría distintos problemas y haría este código impredecible ante cualquiera de estos.

Por esto existen las variables *flag, flag2 y mtolat*, que nos marcarán que el movimiento anterior se ha hecho hacia la izquierda, derecha o en cualquiera de los dos sentidos, respectivamente. A parte de estas variables también se incluye en la pregunta para saber qué paso debemos tomar a los límites de la matriz y así nos aseguramos de que no haya desborde de la misma a la hora de hacer las preguntas. Como se puede apreciar han sido consideradas todas las posibilidades que nos puedan llevar a error e incluidas posteriormente, dándole más robustez al código.

Una vez que a la variable se le ha asignado

el número correspondiente para actuar dependiendo del camino procederemos al siguiente módulo el ACTUADOR DE CAMINOS que viene descrito a través del diagrama de flujo de la siguiente página.

En este diagrama podemos apreciar que la estructura condicional que se maneja es un case-switch el cual se basa en el valor de la variable VAR para actuar en consecuencia. Dependiendo del valor asignado anteriormente se comenzará a desarrollar una acción u otra.

El hecho de que se haya usado una estructura case-switch radica en que, si, por ejemplo, hubiésemos usado una estructura if-else anidada podrían producirse errores de computación, ya que en caso de que la sucesión de pasos sea consecutiva esto produciría que en el mismo bucle while se entrase dos veces lo que nos terminaría dando problemas de bucles infinitos, por ello, optamos por esta doble estructura condicional, la cual nos ha dado muy buenos resultados, como comprobaremos posteriormente.

El modo de funcionamiento se puede apreciar a la perfección en el diagrama de flujo, pregunto cuál es el número que se le ha asignado a la variable VAR en nuestro módulo anterior y comienzo a realizar los cambios pertinentes.

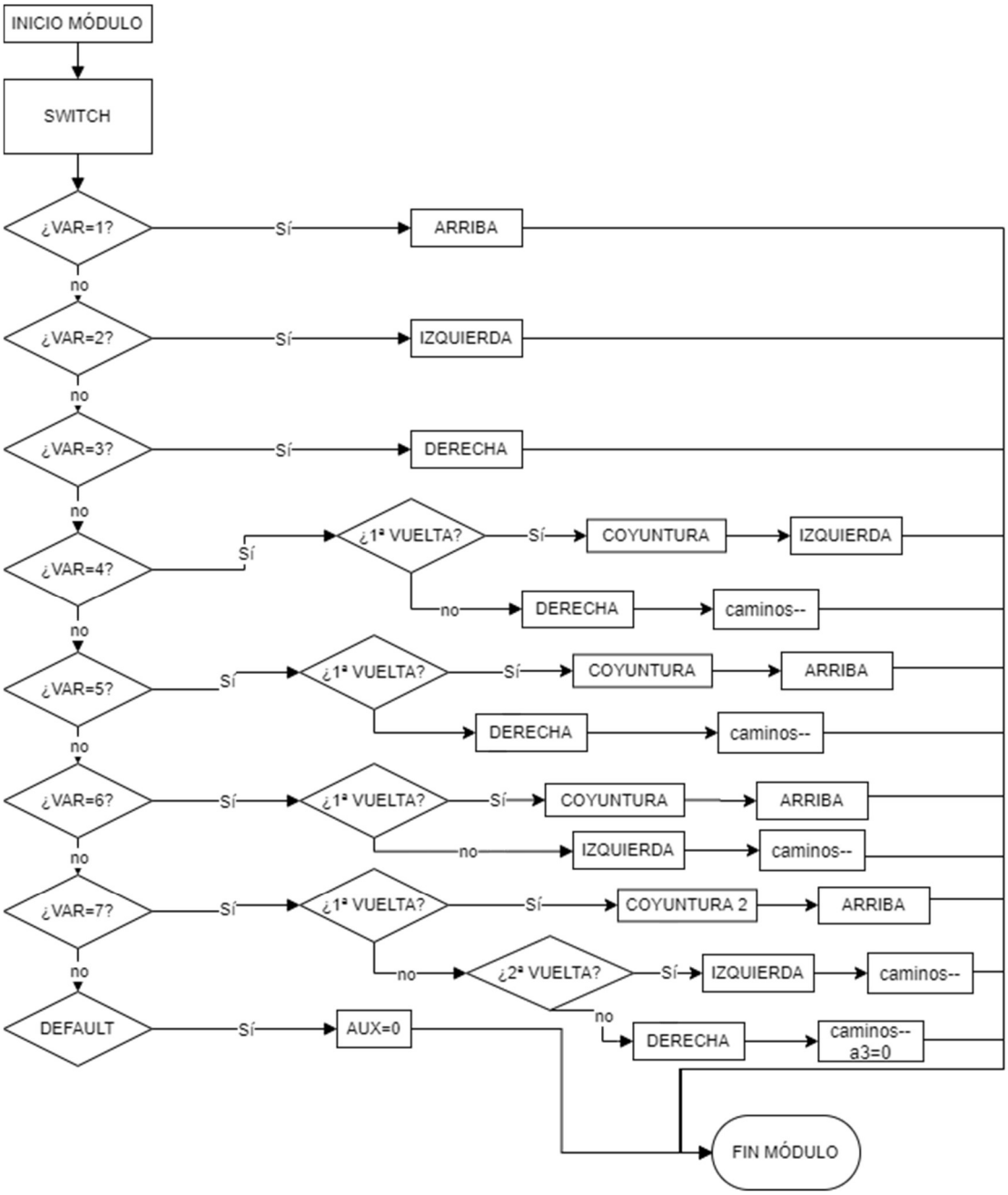
En caso de que el valor asignado sea 1, 2 o 3 nuestro código mandará a nuestro robot hacia adelante, izquierda y derecha, respectivamente con los módulos ARRIBA, IZQUIERDA y DERECHA, que describiremos con posterioridad, una vez completado cualquiera de estos 3 pasos el switch terminará y volveremos al inicio del bucle para saber si debemos hacer otra pasada o terminar el mismo.

La cosa cambia cuando se le asigna a VAR los valores 4,5 o 6 ya que existe un punto de bifurcación en el siguiente paso a tomar, se tomará un punto de coyuntura y se comenzará la ejecución de su parte en el switch. Dentro de éste realizará una pregunta, que será si estamos en la primera vuelta o no, esto se hará a través del valor de una variable que actualizaremos en

función de si hemos pasado por este punto de coyuntura con anterioridad, esta pregunta se presentará posteriormente con su diagrama de flujo asociado.

Una vez realizada dicha pregunta sólo deberemos tomar el camino y la acción asignada para este caso. En caso de que sea la primera vez que encontramos este punto de coyuntura la primera acción a realizar será almacenarlo a través de la función coyuntura, que después discutiremos en profundidad. Una vez que hemos almacenado esta posición deberemos hacer avanzar a nuestro robot hacia la posición indicada por el valor de su variable VAR.

## DIAGRAMA DE FLUJO DE LAS ACCIONES DE CADA CAMINO




En caso de que sea la segunda vuelta que damos deberemos avanzar hacia donde debamos y restar uno a la variable caminos para así demostrar que hay un camino menos que

falte por recorrer y acabar así con la ejecución de esta rama. Los caminos que deberemos tomar en función de que sea la primera vuelta o no y de la variable VAR son:

- VAR=4. Primera vuelta, IZQUIERDA. Segunda vuelta, DERECHA.
- VAR=5. Primera vuelta, ARRIBA. Segunda vuelta, DERECHA.
- VAR=6. Primera vuelta, ARRIBA. Segunda vuelta, IZQUIERDA.

Así, tendríamos cubiertas todas las posibilidades, excepto la última, y la más difícil, la trifurcación, asociada al valor de VAR=7. Ésta se produce cuando en el siguiente paso a abordar existe la posibilidad de ir hacia adelante, la izquierda y a la derecha. En este caso, debemos ser bastante más precavidos que en los anteriores pues la condición de primera, segunda o incluso tercera vuelta, en este caso, varía ya que no la podemos establecer de una forma binaria, lo que nos complica bastante la situación.

Sin embargo, hemos conseguido resolverla de una forma muy sencilla a través del módulo coyuntura 2 y una forma diferente de abordar el módulo 1ª vuelta. Debido a su situación especial, cuando ésta se produzca no se almacenará un punto de coyuntura, sino dos, esta decisión se basa en que realmente este punto en realidad pueden ser dos puntos superpuestos, de forma que necesitan de un tratamiento especial. De todas formas, la forma de comprobar cómo saber si estamos en la primera, segunda o tercera vuelta se explicará con posterioridad al igual que la función coyuntura 2.

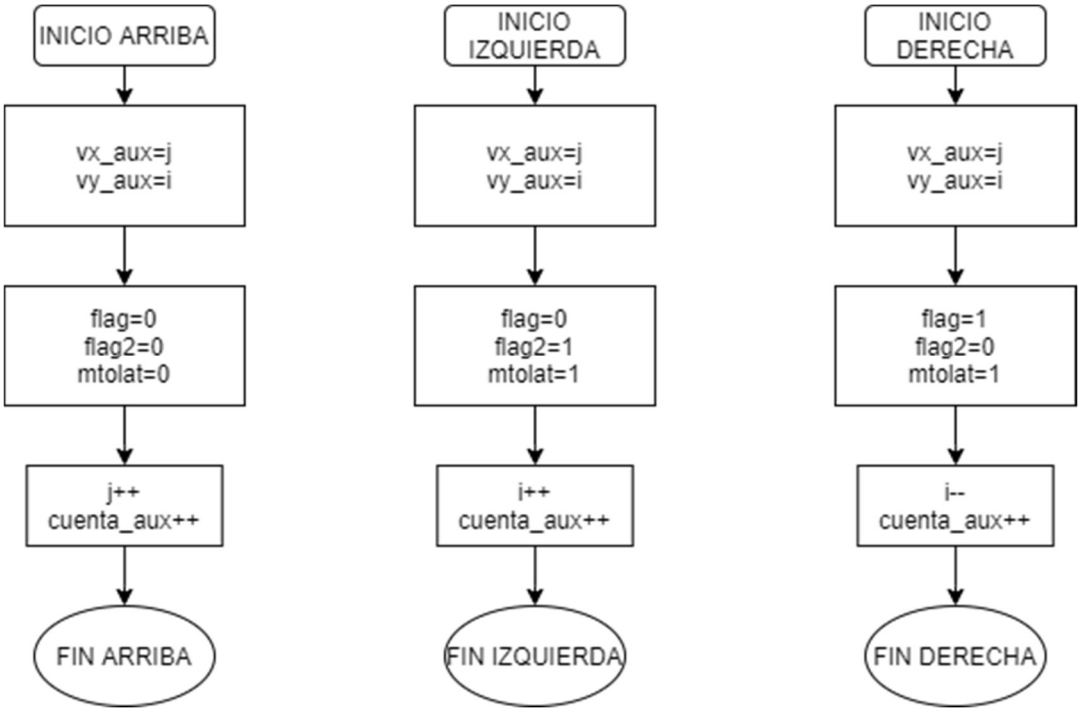
En la primera vuelta de este caso, una vez cumplidas las condiciones el primer camino escogido será el de la función ARRIBA, una vez que se haya terminado este comenzaremos con el módulo IZQUIERDA, y, posteriormente, el último camino por recorrer será hacia DERECHA, así se dará por concluido el camino. Después de aplicar el camino IZQUIERDA y DERECHA restaremos 1 a la variable caminos por cada uno de los completados, y en el último caso además reiniciaremos la variable a3 a 0 de forma que nos permita contemplar el hecho de que exista más de una trifurcación en la superficie a recorrer.

Así, queda descrito el grueso de nuestro programa, el siguiente paso consiste en explicar los pequeños módulos que consiguen hacer que nuestro programa funcione correctamente y nos demuestran que este algoritmo es capaz de adaptarse a cualquier disposición de obstáculos que nos encontremos.

En primer lugar, hablaremos de los módulos *ARRIBA, DERECHA e IZQUIERDA*, estos módulos son muy parecidos entre ellos, con la diferencia de que uno el primero hará que nuestro robot siga hacia adelante, a la izquierda o a la derecha, respectivamente.

El esquema general de cualquiera de estos 3 módulos consiste en:

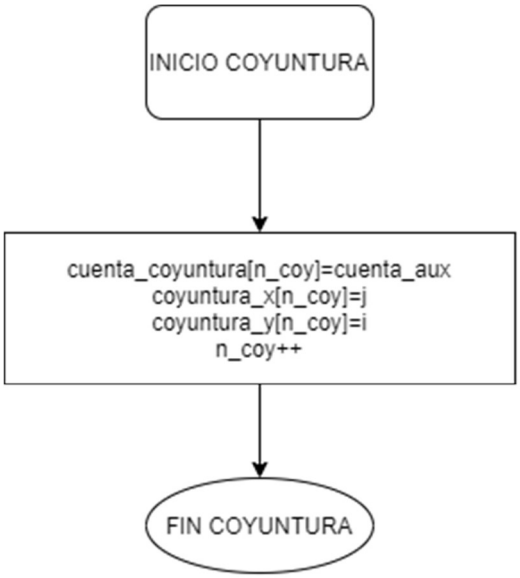
1. Almacenar su posición instantánea en nuestro vector de camino auxiliar.
2. Como ya hemos explicado con anterioridad, se actualizarán las *variables bandera* de movimiento anterior dependiendo de qué módulo sea como se muestra en el diagrama de flujo de la página siguiente.
3. Se aumentará el valor *i* o *j* en función del camino que se tome (siendo la j las filas e i las columnas. También actualizaremos nuestro valor *cuenta_aux* que nos permitirá llevar una cuenta de los pasos que hemos dado en el camino actual.

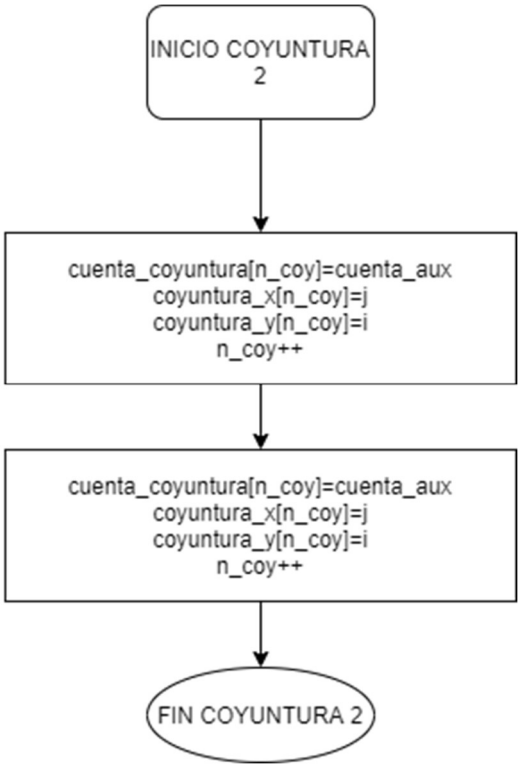



El siguiente módulo que vamos a exponer es el del cálculo de los puntos de coyuntura. Como ya hemos explicado anteriormente un punto de coyuntura no es más que una bifurcación o trifurcación existente en la superficie que hemos anteriormente mapeado, y ahora nos disponemos a recorrer.

Este punto se detecta habiendo preguntado en el módulo selector de caminos por qué puntos de nuestro recorrido están libres para transitar y en caso de que exista más de uno a la vez llegaremos a uno de estos casos. Los casos 4, 5 y 6 tienen bastante en común, mientras que el 7 al ser un poco más distinto es tratado por separado.

Cuando nosotros estamos realizando el recorrido por la matriz y detectamos uno de los puntos de coyuntura nuestra forma de actuar queda descrita por el diagrama de flujo de la siguiente página, el cual detalla los pasos que sigue el procesador de nuestro robot para analizar y tratar un punto de coyuntura.



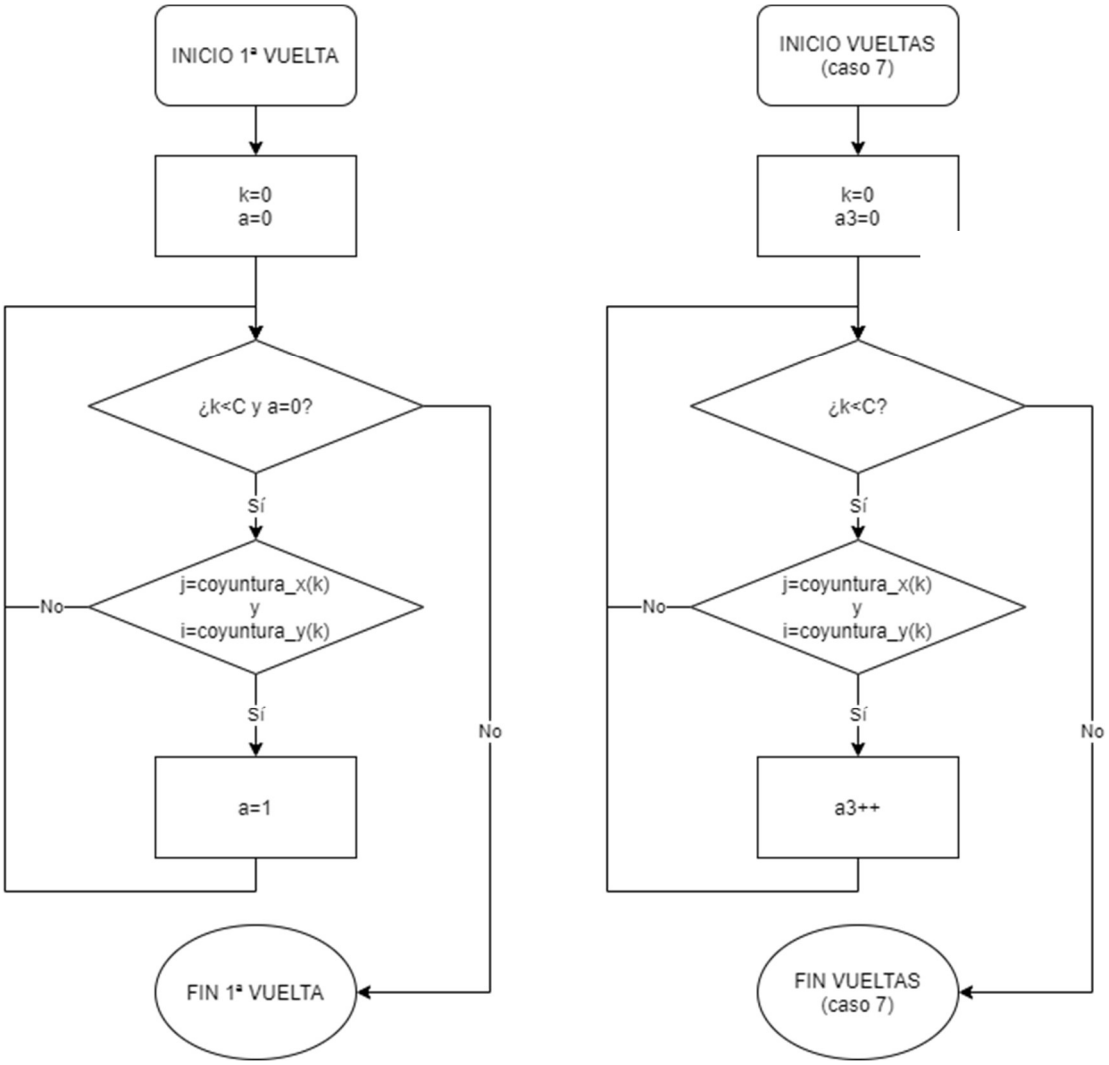




Como podemos apreciar la primera orden que existe es la de almacenar el valor que tenemos en *cuenta_aux* en el vector cuenta_coyuntura para así tener almacenada la cantidad de pasos que habíamos dado con anterioridad de forma que en el momento que se reinicie el recorrido a través de ese punto de coyuntura ahorrarnos los pasos previos donde no existía ninguna bifurcación y así optimizar el procesamiento de datos. Lo mismo haremos con la posición de dicho punto, copiaremos *j* en *coyuntura_x* e *i* en *coyuntura_y*, para así darle a nuestro código la posición de inicio en la segunda vuelta.

Por último, actualizaremos la variable *n_coy*, que nos servirá como índice de nuestros vectores de coyuntura, ya que sólo aumentará cuando existan y detectemos puntos de coyuntura de forma que este índice es específico para esta acción y ninguna otra más.

El diagrama de flujo de la derecha será el módulo de coyuntura para el caso 7, se le llamará coyuntura 2 puesto que en realidad realizamos el módulo coyuntura anteriormente explicado 2 veces de la misma manera, ya que, por motivos computacionales resulta mejor suponer que son como dos puntos de coyuntura distintos en la misma posición. Posteriormente, en el apartado de conclusiones y resultados mostraremos como varían los resultados en función de si tomamos un punto de coyuntura o dos en el caso 7.

El último pequeño módulo que queda por explicar es el que se ocupa de comprobar si nos hallamos o no en la primera vuelta para los casos 4, 5 y 6, y si nos hallamos en la primera, segunda o tercera vuelta en el caso 7. La forma de operar se detalla en el diagrama de flujo de la siguiente página.




En el primer caso (diagrama de la izquierda), podemos comprobar que nuestro algoritmo lo que hace es una estructura iterativa de tipo for, donde recorre los vectores de coyuntura x e y para saber si ya han sido almacenadas esas posiciones de forma que si lo han sido sería nuestra segunda vuelta y si no, la primera.

En caso de que el punto de coyuntura haya sido registrado con anterioridad nos hallaremos en la segunda vuelta de nuestro punto de coyuntura, por tanto, tendremos que indicar a nuestro código que ya lo hemos recorrido, de esta forma, pondremos la variable a igual a 1. Así, nuestra forma de preguntar dentro de nuestro algoritmo si es la primera o la segunda vuelta será comprobando si la variable a está a 0 o a 1, respectivamente. Además, en caso de que la variable a se ponga a 1, el bucle for termina al instante.

En el supuesto de que nos hallemos en el caso 7, la forma de abordar el problema es ligeramente distinta, como podemos ver en el diagrama de flujo anteriormente presentado, también recorreremos nuestro vector de coyuntura de inicio a final para encontrar si esa posición está en el vector anteriormente nombrado. En caso de que no esté se seguirá la ejecución del bucle hasta que termine y en ese caso habremos demostrado que nos encontramos en la primera vuelta. En caso de encontrarnos con el punto en lugar de poner una variable a 1 lo que haremos será aumentarla, y puesto que no poseemos la orden de que en caso de que a3 sea distinto de 0 no se debe seguir, en realidad nos encontraremos con que existen dos puntos de coyuntura guardados en la matriz por lo que hemos explicado con anterioridad.

Por tanto, para saber en qué vuelta nos hallamos preguntaremos, una vez haya terminado el bucle si a3 es igual a 0 (primera vuelta), a 2 (segunda vuelta) o más (tercera vuelta). Esta variable no podrá aumentar una vez llegada a la tercera vuelta ya que una vez que acabe ésta nosotros pondremos a3=0 para asegurarnos que no se producen problemas de desborde en nuestra ejecución.

Llegados a este punto podemos afirmar que este código ha sido explicado en su integridad. Si se poseen dudas de algún tipo se adjuntará el código y una table explicativa de todas las variables que intervienen en nuestro programa con intención de hacer más fácil la comprensión del mismo programa. A la vez, es necesario añadir que todos los diagramas de flujo y el mismo código presentado anteriormente están creados en su integridad por mi persona, siendo este código, por tanto, completamente original.

Sin embargo, este código no aporta la autonomía que se espera de, como se lleva hablando a lo largo de todo el TFG, un robot autónomo. Este programa simplemente resolverá el camino más corto en una superficie dada, por tanto, ahora, se debe obtener la superficie por la que circulará.

Esto se conseguirá por medio de un mapeo previo haciendo uso del sensor de proximidad. El diagrama de flujo que expondrá la forma que se ha decidido a seguir para resultar este mapeo de la superficie puede parecer un tanto lioso, sin embargo, una vez expuesto su comprensión será mayor.

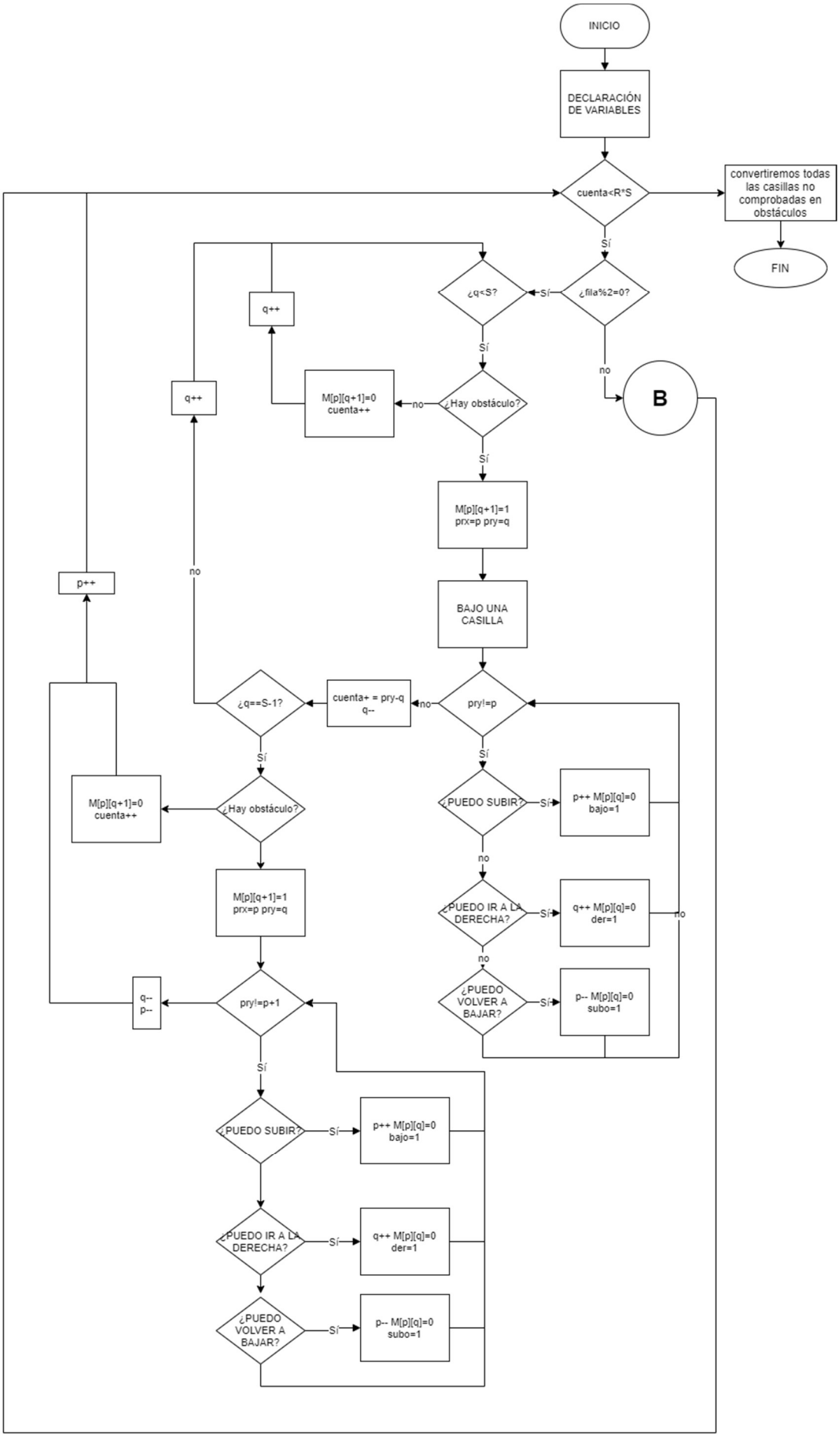
En primer lugar, al programa se le proporciona una matriz rellena de 2 que deberá convertir en una con 0 y 1 en función de si no hay o existe obstáculo en la siguiente posición a ocupar, respectivamente.

Como se puede apreciar en el diagrama de flujo de la siguiente página se comenzará un bucle que vaya recorriendo todas y cada una de las filas de esta matriz, con la singularidad de que la condición de fin de este bucle será que el autómata haya pasado por todas las zonas de la matriz, a través de la variable cuenta.

Una vez que se entra en este bucle, se hace la pregunta de si esa fila es par o impar, en caso de que la fila sea par (ó 0) el robot se moverá de izquierda a derecha, y en el caso contrario, de derecha a izquierda, siguiendo un camino serpenteante, para garantizar la continuidad del mapeo. Debido a que la resolución que se ha decidido adoptar será simétrica para un lado u otro, se ha optado por explicar solamente el caso en el cual la fila es par, el círculo marcado con la B del diagrama de flujo simbolizará la solución en caso de que la fila sea impar.

Ya aclarado este punto, se retoma la explicación de este código de mapeo. El siguiente paso será comenzar a recorrer cada una de las filas preguntando al sensor si existe obstáculo o no. Puesto que no se dispone de un sensor GPS para posicionar al robot, se adoptará guiarlo a través de la odometría, para ello en lugar de hacerle avanzar hasta encontrar un obstáculo, se hará al sistema avanzar una casilla y cada vez que cambie compruebe si la siguiente está ocupada o no.

En caso de que cuando se deba tomar la siguiente casilla no exista impedimento alguno, se almacenará esa posición como 0 (libre) y se avanzará a esta, sumándose también una unidad al contador *cuenta*. En caso contrario, la decisión que se toma es almacenar esa posición con un 1 (obstáculo) y sortearlo. Para ello, se deben realizar una serie de suposiciones para facilitar este mapeo, ya que es un medio para demostrar que el código presentado antes, funciona correctamente de forma automática.


INICIO
DECLARACIÓN DE VARIABLES
cuenta<R*S
convertiremos todas las casillas no comprobadas en obstáculos
FIN
Sí
¿fila%2=0?
¿q<S?
Sí
q++
no
B
Sí
¿Hay obstáculo?
no
M[p][q+1]=0 cuenta++
q++
Sí
M[p][q+1]=1 prx=p pry=q
BAJO UNA CASILLA
pry!=p
no
cuenta+ = pry-q q--
¿q==S-1?
no
p++
Sí
¿PUEDO SUBIR?
Sí
p++ M[p][q]=0 bajo=1
no
¿PUEDO IR A LA DERECHA?
Sí
q++ M[p][q]=0 der=1
no
¿PUEDO VOLVER A BAJAR?
Sí
p-- M[p][q]=0 subo=1
no
¿Hay obstáculo?
M[p][q+1]=0 cuenta++
M[p][q+1]=1 prx=p pry=q
pry!=p+1
q-- p--
Sí
¿PUEDO SUBIR?
Sí
p++ M[p][q]=0 bajo=1
¿PUEDO IR A LA DERECHA?
Sí
q++ M[p][q]=0 der=1
¿PUEDO VOLVER A BAJAR?
Sí
p-- M[p][q]=0 subo=1

Estas suposiciones son:

- Cuando el sistema detecta un obstáculo, no puede existir otro situado bajo la posición del robot.
- Siempre habrá un camino que tomar.
- No se podrán realizar mapeos de estructuras superficiales cerradas, es decir, laberintos y otras variantes.
- El robot solamente podrá avanzar hacia abajo, hacia arriba y hacia la dirección en la que estaba avanzando en ese momento, por tanto, no podrá retroceder para sortear obstáculos.
- En la fila de inicio no podrá existir obstáculos.

Una vez mostradas las suposiciones que se han debido realizar para la resolución del mapeo, se seguirá exponiendo el código de mapeo.

Habiendo encontrado el obstáculo, como se ha comentado en la pasada página, el robot deberá descender una posición para comenzar el movimiento evasivo de dicho obstáculo. A su vez, se almacenará la posición del autómata previa al obstáculo para dos acciones diferentes, la primera, la coordenada y servirá como condición de fin del bucle que se empezará a continuación, de forma que cuando se consiga llegar a la fila en la que se hallaba se considerará terminado el bucle, y el obstáculo sorteado; la segunda acción será actualizar el valor de cuenta sabiendo cuantos casillas se ha desplazado el robot de forma horizontal una vez terminado el movimiento.

De esta forma, después de haber descendido una posición, y haber activado una variable que marcará cuál ha sido el último movimiento realizado para evitar retrocesos, comenzaremos el bucle para esquivar el obstáculo. En este bucle, en primera instancia, se preguntará si el robot puede volver a subir, ya que es el objetivo prioritario para terminar de evadir dicho obstáculo, pero como se halla activa la variable que marca el movimiento anterior, no subirá. Así, una vez que haya preguntado si puede subir y se lo deniegue, el robot intentará ir hacia la izquierda o hacia la derecha en función de la fila en la que se hallase antes, y si tampoco puede avanzar en la dirección determinada, el robot volverá a bajar para sortear el obstáculo encontrado, debido a las suposiciones impuestas anteriormente, no existirá una alternativa diferente y siempre se podrá completar el mapeo.

Una vez que se haya conseguido sortear el obstáculo el bucle de fila volverá al lugar donde lo dejó, y continuará la ejecución del programa. Un caso especial ocurrirá cuando al cambiar de fila exista un obstáculo. Será la última parte del diagrama de flujo, muy parecida a la anteriormente presentada, con la salvedad de que al detectar el obstáculo cuando le toque subir no descenderá, si no que intentará subir, como no podrá hacer esto, se verá obligado a girar hacia el sentido que deba y volver a comprobar si existe algún obstáculo, y así hasta que consiga ascender hasta donde deba.

El código seguirá iterando y resolviendo la incógnita de si existe o no obstáculo en cada casilla hasta que llegue al final de la matriz, una vez que se alcance este punto, se considerará que el mapeo está terminado y comenzará un temporizador para volver a colocar al robot en la posición inicial y recorra la superficie por el camino más corto.

- ***Implementación***

En este apartado mostraremos la forma que se usará para implementar el software que hemos diseñado en la plataforma Code Composer Studio de Texas Instruments v9. El robot, hecho con anterioridad, posee una rutina de inicio y de control de las 4 ruedas del mismo, donde usaremos timer y PWM. Asimismo, se programarán los GPIO para el control de los dos sensores que se usarán para que sea capaz de sortear los obstáculos

En primer lugar, hablaremos de los timers del robot, que serán usados para el control del avance del robot en nuestro mapa:

1. Timer 0: este timer será usado para controlar el tiempo que estará girando el robot para cambiar de dirección.
2. Timer 1: el siguiente timer será usado para controlar el tiempo que las ruedas avanzarán hacia adelante.
3. Timer 2: éste tendrá la utilidad de medir la cantidad de tiempo que se tarda en recibir la señal previamente enviada por el sensor de proximidad.

El Timer1 saltará por medio de una rutina de interrupción que ha sido implementada en el código, la cual solamente realizará la acción de incrementar un contador. Debido al uso que se ha pretendido dar a la interrupción, se optará por reiniciar dicho timer y contador una vez que cada movimiento haya sido efectuado. Es decir, cuando se desee avanzar hacia adelante dejaremos el timer1 corriendo hasta que salte la interrupción por primera vez, y una vez que salga de la condición se reseteará el timer y el contador, así se podrá lograr una mayor precisión en la transición de tiempos.

La rutina implementada en el Timer2 será simple, se programará una interrupción que sea llamada cuando se active y cuando se apague el GPIO, para ello se usará una variable bandera, llamada *echo,* que servirá para indicar si la ejecución se halla en el momento del flanco se subida, o en contraposición, si se halla en el flanco de bajada.

La forma en la que se ha decidido controlar el sensor de proximidad es la siguiente:

1. Cuando se quiera comprobar si la distancia a la que se halla el robot del obstáculo es inferior del límite se llamará a la función *medirdistancia.*

2. Esta función tendrá la tarea de, según como describe el datasheet del HC-SR04, preparar el timer2 que se activará y desactivará cuando el GPIO conectado al pin Echo del sensor se encienda y se apague, respectivamente.

3. Una vez que el timer2 se ha configurado, se envía a través del GPIO conectado a Trig una señal durante 10 microsegundos, que será la que, posteriormente, hará que el sensor envíe 8 pulsos de ultrasonido que serán los que el receptor recibirá una vez hayan rebotado en el obstáculo.

4. Dando una espera adecuada a esta función para que el ultrasonido salga y vuelva, ya se habrá almacenado en memoria el tiempo que habrá transcurrido, y por medio de la fórmula obtenida en el apartado 3 de este documento, esta función devolverá la distancia a la que el autómata se halla del obstáculo.

Por otro lado, otro módulo del DSP que se usará será el PWM, que moverá las ruedas como se le ordene en cada determinado momento., y lo hará a través de distintas funciones creadas por el usuario:

```
void reposo (void)
void adelante0(void)
void adelante1(void)
void adelante2 (void)
void atras(void)
void derecha(void)
void derecha_adelante(void)
void izquierda_adelante(void)
void izquierda(void)
void izquierda_atras(void)
void derecha_atras(void)
void media_vuelta(void)
```

Cada una de estas funciones integrará una forma diferente de combinar el movimiento de las 4 ruedas para conseguir el desplazamiento deseado. A pesar de la diversidad de movimientos integrados en este robot, se ha optado por usar solamente la función media vuelta y la función adelante2 que, en principio, valdrían para la resolución del problema planteado.

Además de los dos módulos comentados anteriormente también se realizará una selección de los GPIO que serán usados por el sistema. Por último, una de las funciones integradas en la memoria del robot será la desarrollada unas páginas atrás, con el actual nombre de `int encuentra_camino_mas_corto(int, int)`. Esta función devolverá un entero que será la variable cuenta que indica la cantidad de pasos que serán necesarios para recorrer el mapa de la forma más rápida.

Cuando ya ha sido llamada esta función, se poseen entonces dos vectores que dirán las posiciones que dictan el camino más rápido al autómata, por tanto, en el bucle infinito dentro del while, se deberá implementar un pequeño código que marque por donde se debe mover el robot en función de los dos vectores anteriores *vx* y *vy.*

El código tendrá la función de comprobar cuál será el siguiente paso a tomar, y, en consecuencia, y dependiendo del paso anterior que se haya realizado, dará la orden de avanzar, girar a la izquierda o girar a la derecha para completar el camino que se deba hacer.

Para añadir una funcionalidad extra, aportando así mayor autonomía al robot en cuestión, se ha decidido implementar una función que, en caso de que en el apartado de recorrer la matriz buscando el camino más corto exista un obstáculo no registrado, conseguir llegar al final sorteando dicho obstáculo.

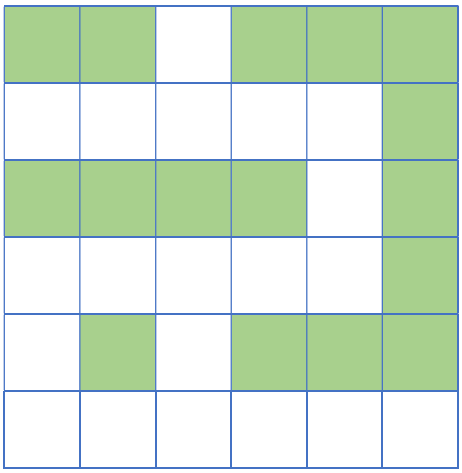
La forma de resolución que se ha decidido tomar para conseguir esto se basa en preguntar en cada momento que se avance si la distancia a la que se halla el robot del obstáculo es inferior al límite, si no es así, se seguirá el transcurso del programa planteado anteriormente. En caso de que el autómata se halle a menos de la distancia límite con el obstáculo, se procederá a parar la ejecución del programa y llamar a la función `encuentra_camino_mas_corto` dándole los parámetros *x* e *y* que indican la posición actual, haciendo así, que se calcule una nueva ruta desde ese punto hasta el final, habiendo añadido además el nuevo obstáculo al mapa para calcular el camino más corto.

Una vez que se ha implementado todo el proyecto de CCS, habiendo incluido las librerías necesarias, así como los ficheros de los periféricos que se usarán, y solucionados todos los errores de implementación, se procederá a grabarlo en la memoria del robot para así comenzar a obtener los resultados finales de este proyecto.

# 5. Resultados y conclusiones

En este apartado final del proyecto se expondrá el funcionamiento del autómata para distintos casos de funcionamiento, y así mostrar la utilidad del mismo.

En primer lugar, se tratará el caso en el cual se le cede una matriz al robot que ya posee los obstáculos localizados en la misma, es decir, se le da una matriz que ya está rellena de 0 y 1, por tanto, lo único que deberá hacer el sistema será encontrar el camino más corto y recorrerlo. En este caso no se realizará mapeo previo para mostrar cómo funciona el código por sí solo. La siguiente, será la matriz que se ha grabado en la memoria flash del robot y que éste deberá recorrer:



De esta forma, como se ha expuesto anteriormente, en primer lugar, la función para encontrar el camino más corto actuará, obteniendo el valor de cuenta, así como los vectores vx y vy que marcarán el camino que debe seguir el robot para completar la prueba.

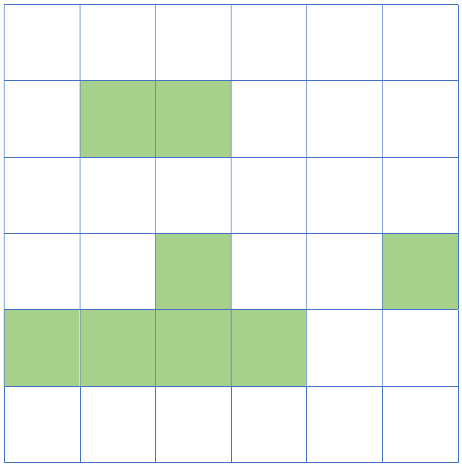
Por tanto, el recorrido que deberá hacer el autómata será:

1. Avanzar hacia adelante un cuadrado.
2. Girar a la izquierda y avanzar un cuadrado.
3. Avanzar dos cuadrados hacia la izquierda (hacia adelante para el robot).
4. Girar a la derecha y avanzar un cuadrado.
5. Avanzar dos cuadrados hacia adelante.
6. Girar a la derecha y avanzar un cuadrado.
7. Avanzar dos cuadrados hacia la derecha
8. Girar a la izquierda y avanzar un cuadrado
9. Avanzar dos cuadrados hacia adelante
10. Una vez que ha llegado al final se queda parado.

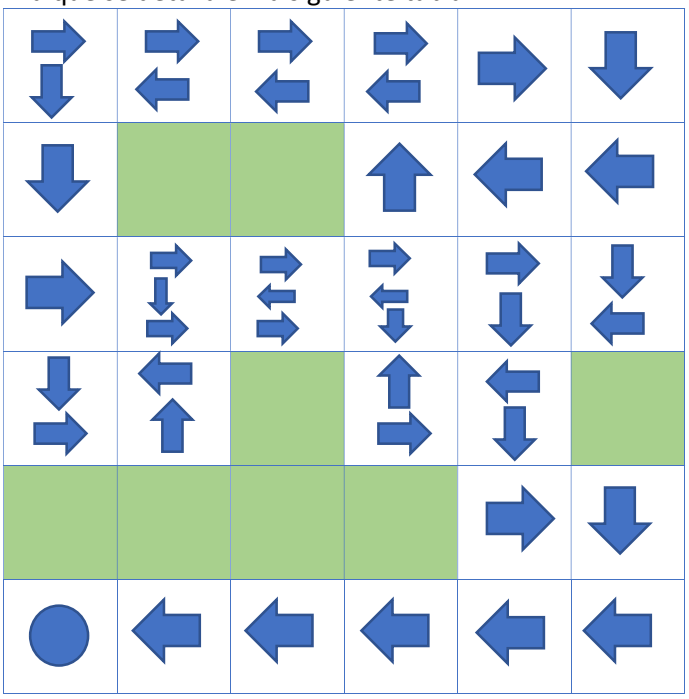
A pesar de los pequeños desajustes debidos en gran parte a los errores sistemáticos, se puede comprobar que el experimento ha sido un éxito, ya que el robot es capaz de desplazarse a lo largo de la matriz sin ningún problema, como se verá en el vídeo demostrativo de la exposición.

Una vez que se ha demostrado que el código funciona, se procederá a realizar las comprobaciones con un mapeo previo de la superficie, dándole así más independencia al sistema, ya que el mismo se encargará de localizar todos los obstáculos que existan.

En este caso, se le cederá al robot una matriz repleta únicamente de 2, y en función de los obstáculos que detecte colocará un 0 o un 1. De esta forma, el mapa real que deberá recorrer el autómata será el siguiente, siendo oscurecida la casilla en la que exista un obstáculo real:



De esta forma se colocará al robot en el inicio (0,0) de dicha matriz y comenzará a recorrerla de la forma que se detalla en la siguiente tabla:



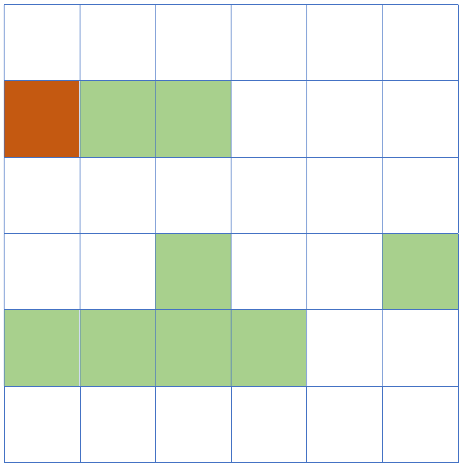
Una vez realizado este mapeo, mostrado por el recorrido anterior, el sistema procede a calcular el camino más corto, a la vista de esta superficie. El camino escogido por el sistema será el siguiente:

1. (0,0)
2. (1,0)
3. (2,0)
4. (2,1)
5. (2,2)
6. (2,3)
7. (3,3)
8. (3,4)
9. (4,4)
10. (4,5)

Aunque existe otro camino con el mismo número de pasos, se escoge este debido a que éste fue seleccionado con anterioridad, y, por tanto, se mantiene el primer camino escogido. De esta forma, por medio de la programación introducida en CCS el robot será capaz de recorrer sin problema el camino elegido por el código con las siguientes indicaciones:

1. Avanzar hacia adelante dos cuadrados.
2. Girar a la izquierda y avanzar tres cuadrados.
3. Girar a la derecha y avanzar un cuadrado
4. Girar a la izquierda y avanzar un cuadrado.
5. Avanzar dos cuadrados hacia adelante.
6. Una vez llega al final se queda parado.

El tercer caso experimental que se propondrá estará basado en el mismo mapa de obstáculos presentado en el caso anterior, de forma que el camino más corto escogido será también el miso que se obtuvo en el mismo último caso. Sin embargo, en este caso, después del mapeo se introducirá un obstáculo adicional en el cuadrado con posición (1,0), de forma que el itinerario se verá alterado según lo previsto, yendo en este caso por el segundo camino posible. La forma del mapa resultante tras el mapeo será la siguiente:



El cuadrado rojo representará el obstáculo colocado tras el mapeo, y el itinerario que siguió el robot es el siguiente:

1. (0,0)
2. (0,1)
3. (0,2)
4. (0,3)
5. (1,3)
6. (2,3)
7. (3,3)
8. (3,4)
9. (4,4)
10. (4,5)

En este caso el camino a seguir cambió desde el inicio, sin embargo, no tiene porqué se así, ya que cada vez que el robot cambia de casilla comprueba si en el siguiente en su itinerario existe o no obstáculo, si no hay ningún impedimento a seguir el transcurso, seguirá el camino que debía seguir; en caso afirmativo, el sistema volverá a llamar a la función para calcular la ruta dándole la ubicación actual. Por ello, si el obstáculo se hubiese colocado en la posición (3,3), lo habría sorteado y habría trazado la ruta más corta hasta el final del recorrido.

Llegados a este punto, se ha conseguido demostrar el funcionamiento del robot de una forma autónoma a través de una superficie con obstáculos en un entorno estático, habiendo hecho un mapeo previo también.

Es necesario añadir que, este Trabajo de Fin de Grado tiene muchas posibilidades de ampliación, como, por ejemplo, introducir que el sistema de posicionamiento del coche este regulado por un GPS para garantizar su buena localización en entornos de grandes dimensiones, y así, no depender en tanta medida de la odometría; Asimismo, también conseguir que el mapeo realizado por el robot se almacene en una tarjeta de memoria y no se borre cuando éste se apague, para conseguir de ahí en adelante que el robot llegue de un punto a otro sin necesidad de hacer un mapeo previo todas las veces que se desplace. Esto son solamente dos ejemplos de todas las ideas que pueden surgir a cualquier persona que esté trabajando en estos temas.

En conclusión, el aprendizaje adquirido tras la realización de este Trabajo de Fin de Grado ha servido para reforzar mis conocimientos mediante la metodología de ensayo-error. Los obstáculos planteados por cada fase de este proyecto han supuesto un reto que poco a poco y con trabajo he conseguido alcanzar.

Considero que este trabajo, junto a los años de trabajo en la carrera de GITI han construido una sólida base de conocimientos en el mundo de la ingeniería, que confío me den el paso a convertirme en Graduado, y en un futuro, en un auténtico ingeniero.

# 6. Bibliografía

[1] Artificial intelligence: A modern approach: Stuart Russell and Peter Norvig, (Prentice Hall, Englewood Cliffs, NJ, 1995)
https://www.sciencedirect.com/science/article/pii/0004370296000070?via%3Dihub

[2] https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial

[3] https://dle.rae.es/robot

[4] https://www.fing.edu.uy/inco/grupos/mina/pGrado/pgSLAM/documentos/eda.pdf

[5] Shortest Paths Algorithms: Theory and Experimental Evaluation: Boris V. Cherkassy, Andrew V. Goldberg, Tomasz Radzik
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.8746&rep=rep1&type=pdf

[6] University of Bristol article
https://web.archive.org/web/20100113034225/http://www.bris.ac.uk/synaptic/research/projects/memory/spatialmem.htm

[7] https://www.news-medical.net/health/Hippocampus-Functions-(Spanish).aspx

[8] Spatial Cells in the Hippocampal Formation: John O'Keefe
https://www.nobelprize.org/uploads/2018/06/okeefe-lecture.pdf

[9] Milford, Michael, Jacobson, Adam, Chen, Zetao, & Wyeth, Gordon (2016) RatSLAM : using models of rodent hippocampus for robot navigation and beyond. In Robotics Research: The 16th International Symposium ISRR [Springer Tracts in Advanced Robotics, Volume 114], Springer, Singapore, pp. 467- 485.
https://www.researchgate.net/publication/301591193_RatSLAM_Using_Models_of_Rodent_Hippocampus_for_Robot_Navigation_and_Beyond

[10] Localización y mapeado en entornos de oficina
https://rua.ua.es/dspace/bitstream/10045/9887/4/Gallardo-Lopez-Domingo_3.pdf

[11] https://quantdare.com/filtro-kalman/

[12] Influencia de los parámetros de un filtro de partículas en la solución al problema de SLAM A. Gil, O. Reinoso, Member, IEEE, L. Payá, M. Ballesta
https://arvc.umh.es/documentos/articulos/2007_IEEELA_influencia.pdf

[13] Construcción de Mapas y Localización Simultánea con Robots Móviles. Tesis de Maestría en Ciencias Computacionales Víctor Manuel Jaquez Leal
https://www.ceyusa.com/documentos/tesis/presentacion.pdf

[14] https://www.ti.com/lit/ds/symlink/drv8835.pdf

[15] https://datasheet.octopart.com/TMS320F28335-Texas-Instruments-datasheet-152273.pdf

[16] https://www.dfrobot.com/product-100.html

# 7. Imágenes


1. https://mars.nasa.gov/msl/home/
2. http://www.bbc.co.uk/london/travel/downloads/tube_map.html
3. https://web.archive.org/web/20100113034225/http://www.bris.ac.uk/synaptic/research/projects/memory/spatialmem.htm
4. https://web.archive.org/web/20100113034225/http://www.bris.ac.uk/synaptic/research/projects/memory/spatialmem.htm
5. https://www.researchgate.net/figure/Grid-cells-and-place-cells-Left-A-grid-cell-from-the-entorhinal-cortex-of-the-rat_fig1_272839931
6. https://en.wikipedia.org/wiki/Grid_cell#/media/File:Equilateral_Triangle_Lattice.svg
7. https://www.researchgate.net/figure/Figura-26-Mapas-topologico-y-semantico_fig2_39655692
8. https://es.wikipedia.org/wiki/Grafo#/media/Archivo:6n-graf.svg
9. Hecha por mi
10. Hecha por mi
11. https://datasheet.octopart.com/TMS320F28335-Texas-Instruments-datasheet-152273.pdf
12. Propia
13. Propia
14. Web: https://robotics.nasa.gov/
15. Link: [https://www.nasa.gov/audience/foreducators/diypodcast/robots-index-diy.html]
16. https://www.amazon.es/HC%E4%B8%80SR04-HCSR04-Detector-Ultras%C3%B3nica-Distancia/dp/B07MR5QP16

# 8. ANEXO: Código en C del main.

```
#include "DSP2833x_Device.h"

void (*movimiento)(void);

#define R 6     //Altura de la matriz
#define S 6     //Anchura de la matriz
#define T 20
#define C 50
#define g_izq 100000
#define g_der 300000
#define d_lim 15.00

int vx[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int vy[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};


// Prototipado de funciones externas
extern void InitSysCtrl(void);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);

// Prototipado de funciones internas

//Interrupciones
__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);
__interrupt void obstaculo(void);
//Configuración de pines y PWM
void Setup_ePWM(void);
void Gpio_select(void);
//Funciones de movimiento
void reposo (void);
void adelante0(void);
void adelante1(void);
void adelante2 (void);
void atras(void);
void derecha(void);
void derecha_adelante(void);
void izquierda_adelante(void);
void izquierda(void);
void izquierda_atras(void);
void derecha_atras(void);
void media_vuelta(void);
//Funciones específicas para código
float medirdistancia(int *);
void delay(int);
int encuentra_camino_mas_corto(int , int);
void premapeo (void);

void avanzo(void);
void giro_der(void);
void giro_izq(void);

// Declaración de variables globales y la matriz a recorrer

int M[R][S]={
                {2 , 2 , 2 , 2 , 2 , 2},
                {2 , 2 , 2 , 2 , 2 , 2},
                {2 , 2 , 2 , 2 , 2 , 2},
                {2 , 2 , 2 , 2 , 2 , 2},
                {2 , 2 , 2 , 2 , 2 , 2},
                {2 , 2 , 2 , 2 , 2 , 2} };
```

```
int echo=0;
long int cputime=0 ,cputime1=0 , cputime2=0;
void main(void)

{

    InitSysCtrl();                          // Inicialización del sistema
(DSP2833x_SysCtrl.c):
    Gpio_select();                          // Selección de los GPIO que se usarán
    DINT;                                   // Deshabilito las interrupciones
    Setup_ePWM();                           // Inicialización PWM
    InitPieCtrl();                          // Inicialización de la PIE
(DSP2833x_PieCtrl.c)

    IER = 0x0000;
    IFR = 0x0000;

    InitPieVectTable();                     // Inicialización de la tabla PIE
(DSP2833x_PieVect.c)

    // Modificación de la PIE para direccionar las rutinas de interrupción
    EALLOW;
    PieVectTable.TINT0 = &cpu_timer0_isr;
    PieVectTable.XINT13 = &cpu_timer1_isr;
    PieVectTable.TINT2 = &cpu_timer2_isr;
    PieVectTable.XINT1 = &obstaculo;
    EDIS;

    InitCpuTimers();   // Inicialización de los timer y configuración

   ConfigCpuTimer(&CpuTimer0, 150, 10);             //timer para rotar el coche
   ConfigCpuTimer(&CpuTimer1, 150, 2000000);        //timer para hacer avanzar el
coche
   ConfigCpuTimer(&CpuTimer2, 150, 0xffffffff);     //timer para calcular la
distancia

   IER |= M_INT1;
   IER |= M_INT13;
   IER |= M_INT14;

   PieCtrlRegs.PIEIER1.bit.INTx4 = 1;   // Se habilita la interrupción en la PIE,
INT1.4, fuente 1 (XINT1)
   PieCtrlRegs.PIEIER1.bit.INTx7 = 1;   // Se habilita la interrupción en la PIE,
INT1.7, fuente 1 (CPUTTIMER0)

   XIntruptRegs.XINT1CR.bit.POLARITY = 3;  // Interrupción generada por flancos de
subida y bajada
   XIntruptRegs.XINT1CR.bit.ENABLE = 1;    // Habilitamos la interrupción externa
XINT1

   EINT;                                   // Se habilitan todas las lÃneas de
interrupciÃ³n
   ERTM;                                   // Enable Global realtime interrupt DBGM

//Declaración de variables locales
    int i ,der=0, izq=0;
    int N;
    float dist;
    int x=2;
    int y=0;

//Llamamos a la función de premapeo
   premapeo();
//Llamamos a la función para recorrer el camino más corto
   N=encuentra_camino_mas_corto(x,y);
```

```
while(1)
{
   for(i=0 ; i<N ; i++)
    {

            if(y+1==vx[i+1])
           {
                y++;

               if(der==1)
               {
                   giro_der();
                   der=0;
               }
               else if(izq==1)
              {
                   giro_izq();
                   izq=0;
              }

               dist=medirdistancia(&echo);
               if ( dist < 20 )
                {
                    M[y+1][x]=1;
                    N=encuentra_camino_mas_corto(x,y);
                    i=-1;
                }
                else
                    avanzo();
           }

           else if(x+1==vy[i+1])
           {
               x++;

               if(izq==0)
               {
                   giro_izq();
                   izq=1;
               }
               dist=medirdistancia(&echo);
               if ( dist < 20 )
               {
                   M[y][x+1]=1;
                   N=encuentra_camino_mas_corto(x,y);
                   i=-1;
               }
               else
                   avanzo();
                   der=0;
            }
           else if(x-1==vy[i+1])
           {
              x--;

               if(der==0)
               {
                   giro_der();
                   der=1;
               }
               dist=medirdistancia(&echo);
               if ( dist < 20 )
               {
                   M[y][x-1]=1;
```

```
                        N=encuentra_camino_mas_corto(x,y);
                        i=-1;
                    }
                    else
                        avanzo();
                        izq=0;
                 }
                else
                {
                    movimiento=reposo;
                    movimiento();
                }

        }
           while(1)
           {
               movimiento=reposo;
               movimiento();
           }
     }

}
```

## 9. Código en C de la función de guiado óptimo

```
int encuentra_camino_mas_corto(int x, int y)
{


    int cuenta=0, cuenta_aux=0, vueltas=0;

    //a nos dice si es la primera o la segunda vuelta
    int a,a3=0,b=0, caminos=0;
    //indices de ejecución
    int i, j,k, l=0, n_coy=0;
    //variable que se activa la primera vez que consigo un camino
    int h=0;

    //variables auxiliares
    //aux nos dice que no hay camino
    //var sirve para seleccionar la acción que toca
    int aux=0, var;

    //banderas del mto anterior
    int mtolat=0, flag2=0, flag=0;

    //posicion de los puntos de bifurcación
    int coyuntura_x[C]={0,0,0,0,0,0,0,0,0,0};
    int coyuntura_y[C]={0,0,0,0,0,0,0,0,0,0};
    //altura de la cuenta cuando se produce un punto de coyuntura
    int cuenta_coyuntura[C]={0,0,0,0,0,0,0,0,0,0};

    //vectores auxiliares y finales de las posiciones del camino
    //int vx[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    int vx_aux[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    //int vy[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    int vy_aux[T]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

    //el índice i nos marcará la columna (vy)
        i=x;
    //el índice j nos marcará la fila (vx)
        j=y;

    do{

    while (j<R-1 && aux==0)
    {
        //selecciono el caso en el que estamos con un if-else
        if (M[j+1][i]==0 && M[j][i+1]==0 && M[j][i-1]==0 && flag==0 && i+1<=S-1 && i-
1>=0 && flag2==0 && mtolat==0)
        {   var=7;  }
        else{
            if (M[j+1][i]==0 && M[j][i+1]==0 && flag==0 && i+1<=S-1)
                {var=6; }
            else{
                if (M[j+1][i]==0 && M[j][i-1]==0 && flag2==0 && i-1>=0 && mtolat==0)
                    {var=5; }
                else{
                    if (M[j][i+1]==0 && M[j][i-1]==0 && mtolat==0 && i+1<=S-1 && i-
1>=0)
                        {var=4; }
                    else{
                        if (M[j+1][i]==0)
                            {var=1; }
                        else{
                            if (M[j][i+1]==0 && flag==0)
                                {var=2; }
                            else{
                                if (M[j][i-1]==0)
```

```
                                {var=3; }
                            else {var=0;    };
                            }
                        }
                    }
                }
            }
        }

    //dependiendo de donde me hallo con un switch digo
    //a mi robot qué es lo que debe hacer

    switch (var)
    {
        case 1:
        {

            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag=0;
            flag2=0;
            mtolat=0;

            j++;
            cuenta_aux++;

            break;
        }
        case 2:
        {

            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag2=1;
            flag=0;
            mtolat=1;

            i++;
            cuenta_aux++;

            break;
        }

        case 3:
        {
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag2=0;
            flag=1;
            mtolat=1;

            i--;
            cuenta_aux++;

            break;
        }
        //IZQUIERDA Y DERECHA
        case 4:
        {

            //compruebo si es la primera vez que detecto este punto o no
            for(k=0, a=0;k<C && a==0;k++)
            {
```

```
            if(j==coyuntura_x[k] && i==coyuntura_y[k])
                a=1;
        }
        //si no, voy hacia adelante
        if (a==0)
        {
            caminos++;
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            //almaceno el estado de cuenta y
            //la posicion del pto de coyuntura
            cuenta_coyuntura[n_coy]=cuenta_aux;
            coyuntura_x[n_coy]=j;
            coyuntura_y[n_coy]=i;
            n_coy++;

            flag2=1;
            mtolat=1;
            flag=0;
            i++;
            cuenta_aux++;
        }
        //si ya he pasado, giro a la derecha
        else{
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag2=0;
            flag=1;
            mtolat=1;

            i--;
            cuenta_aux++;
            caminos--;
            }

        break;
    }
    //ARRIBA Y DERECHA(--)
    case 5:
    {

        //compruebo si es la primera vez que detecto este punto o no
        for(k=0, a=0;k<C && a==0;k++)
        {
            if(j==coyuntura_x[k] && i==coyuntura_y[k])
                a=1;
                //printf(" HOLA %d",k);
        }

        if (a==0)
        {
            caminos ++;
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            //almaceno el estado de cuenta y
            //la posicion del pto de coyuntura
            cuenta_coyuntura[n_coy]=cuenta_aux;
            coyuntura_x[n_coy]=j;
            coyuntura_y[n_coy]=i;
            n_coy++;

            flag2=0;
            mtolat=0;
```

```
            flag=0;
            j++;
            cuenta_aux++;
        }
        else{
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag2=0;
            flag=1;
            mtolat=1;

            i--;
            cuenta_aux++;
            caminos--;
            }

        break;
    }
    //ARRIBA E IZQUIERDA(++)
    case 6:
    {
        //aux=0;
        //compruebo si es la primera vez que detecto este punto o no
        for(k=0, a=0;k<C && a==0;k++)
        {
            if(j==coyuntura_x[k] && i==coyuntura_y[k])
                a=1;
        }

        if (a==0)
        {
            caminos ++;
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            //almaceno el estado de cuenta y
            //la posicion del pto de coyuntura
            cuenta_coyuntura[n_coy]=cuenta_aux;
            coyuntura_x[n_coy]=j;
            coyuntura_y[n_coy]=i;
            n_coy++;

            mtolat=0;
            flag=0;
            flag2=0;

            j++;
            cuenta_aux++;
        }
        else{
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag=0;
            mtolat=1;
            flag2=1;

            i++;
            cuenta_aux++;
            caminos--;
            }
        break;
    }
        //hay una trifurcación hacia arriba, izquierda y derecha
    case 7:
```

```
{
        //compruebo si es la primera vez que detecto este punto o no
    for(k=0; k<C ;k++)
    {
        if(j==coyuntura_x[k] && i==coyuntura_y[k])
            a3++;
    }

    if(a3==0) //voy hacia adelante
    {
        caminos = caminos+2;
        vx_aux[cuenta_aux]=j;
        vy_aux[cuenta_aux]=i;


        //almaceno el estado de cuenta y
        //la posicion del pto de coyuntura
        cuenta_coyuntura[n_coy]=cuenta_aux;
        coyuntura_x[n_coy]=j;
        coyuntura_y[n_coy]=i;
        n_coy++;
        //lo hago por duplicado ya que en realidad son dos puntos
        cuenta_coyuntura[n_coy]=cuenta_aux;
        coyuntura_x[n_coy]=j;
        coyuntura_y[n_coy]=i;
        n_coy++;

        mtolat=0;
        flag=0;
        flag2=0;

        j++;
        cuenta_aux++;
    }
    else
    {
        if(a3==2) //voy a la izquierda
        {
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag=0;
            mtolat=1;
            flag2=1;

            i++;
            cuenta_aux++;
            caminos--;
        }
        else //voy a la derecha
        {
            vx_aux[cuenta_aux]=j;
            vy_aux[cuenta_aux]=i;

            flag2=0;
            flag=1;
            mtolat=1;

            i--;
            cuenta_aux++;
            caminos--;
            a3=0;
        }
    }
    break;
}
```

```
            default:
            {
                aux=1; //no hay camino
                break;
            }
        }

    } //fin del while
    vueltas++;

    if (j>=R-1)
    {           //voy a comprobar si ha llegado al final de mi camino
        b++;    //en alguna ocasión
    }

    if (j>=R-1 && b==1)     //si es la primera vez que termino activo
        h=1;                //la variable para iniciar la cuenta
    else
        h=0;

        //sustituyo cuenta y las posiciones si ésta es menor
        if (cuenta_aux<cuenta && aux==0 || h==1)
        {
            cuenta=cuenta_aux;
            for(k=0;k<cuenta;k++)
            {
                vx[k]=vx_aux[k];
                vy[k]=vy_aux[k];
            }
        }



    //por si sucede que un camino no tenga final pero
    //queden más por recorrer caminos reinicio aux
    aux=0;
    //doy paso para el siguiente punto de coyuntura
    if (caminos>0)
    {
        cuenta_aux=cuenta_coyuntura[l];
        i=coyuntura_y[l];
        j=coyuntura_x[l];
        l++;
    }

    //printf(" caminos: %d\n", caminos+1);


    }   //final del do
    while (caminos>0); //si ya no hay más caminos acabo mi bucle

        //si no hay camino pongo todos mis vectores a 0
    if (b==0)
    {
        //printf("NO HAY CAMINO\n");
        cuenta=0;
        for(k=0;k<T;k++)
        {
            vx[k]=0;
            vy[k]=0;
        }
    }
return(cuenta);}
```

# 10. Código en C de interrupciones y funciones de movimiento

```
void reposo(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500;       //tra der
}
void adelante0(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.7;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.7;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.7;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.7;       //tra der
}

void adelante1(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.5;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.5;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.5;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.5;       //tra der
}

void adelante2 (void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.3;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.3;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.3;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.3;       //tra der
}


void atras(void){

    GpioDataRegs.GPASET.bit.GPIO12 = 1;
    GpioDataRegs.GPASET.bit.GPIO15 = 1;
    GpioDataRegs.GPASET.bit.GPIO6 = 1;
    GpioDataRegs.GPASET.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.5;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.5;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.5;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.5;       //tra der
```

```
}

void derecha (void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.1;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.1;       //tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.9;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.9;       //tra der
}

void derecha_adelante(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.2;      //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.2;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.7;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.7;       //tras der
}

void izquierda_adelante(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.7;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.7;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.2;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.2;       //tras der
}

void izquierda(void){

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.9;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.9;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.1;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.1;       //tras der
}

void izquierda_atras(void){

    GpioDataRegs.GPASET.bit.GPIO12 = 1;
    GpioDataRegs.GPASET.bit.GPIO15 = 1;
    GpioDataRegs.GPASET.bit.GPIO6 = 1;
    GpioDataRegs.GPASET.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.7;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.7;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.2;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.2;       //tras der
```

```
}

void derecha_atras(void){

    GpioDataRegs.GPASET.bit.GPIO12 = 1;
    GpioDataRegs.GPASET.bit.GPIO15 = 1;
    GpioDataRegs.GPASET.bit.GPIO6 = 1;
    GpioDataRegs.GPASET.bit.GPIO16 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.2;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.2;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.7;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.7;       //tras der
}

void media_vuelta(void){
    GpioDataRegs.GPACLEAR.bit.GPIO16 = 1;
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    GpioDataRegs.GPASET.bit.GPIO15 = 1;
    GpioDataRegs.GPASET.bit.GPIO12 = 1;

    EPwm1Regs.CMPA.half.CMPA = 37500*0.3;       //del iz
    EPwm6Regs.CMPA.half.CMPA = 37500*0.3;       // tras iz
    EPwm2Regs.CMPA.half.CMPA = 37500*0.3;       //del der
    EPwm3Regs.CMPA.half.CMPA = 37500*0.3;       //tras der

}
void avanzo (void)
{
    StartCpuTimer1();
    while(CpuTimer1.InterruptCount<1)
     {
         movimiento=adelante2;
         movimiento();
     }
   StopCpuTimer1();
   ReloadCpuTimer1();
   CpuTimer1.InterruptCount=0;
}
void giro_izq(void)
{
     StartCpuTimer0();
     while(CpuTimer0.InterruptCount<g_izq)
        {
            movimiento=media_vuelta;
            movimiento();
        }
     StopCpuTimer0();
     ReloadCpuTimer0();
     CpuTimer0.InterruptCount=0;
}
void giro_der(void)
{
      StartCpuTimer0();
      while(CpuTimer0.InterruptCount<g_der)
         {
             movimiento=media_vuelta;
             movimiento();
         }
      StopCpuTimer0();
      ReloadCpuTimer0();
      CpuTimer0.InterruptCount=0;
}

__interrupt void
cpu_timer0_isr(void)
```

```
{

    CpuTimer0.InterruptCount++;

    //
    // The CPU acknowledges the interrupt.
    //
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

}
__interrupt void
cpu_timer1_isr(void)
{
    EALLOW;
    CpuTimer1.InterruptCount++;

    //
    // The CPU acknowledges the interrupt.
    //
    EDIS;
}
__interrupt void
cpu_timer2_isr(void)
{
    EALLOW;
    CpuTimer2.InterruptCount++;

    //
    // The CPU acknowledges the interrupt.
    //
    EDIS;
}
```

# 11. Código en C para sensor de proximidad

```
float medirdistancia (int *echo)
{
    float microS=0;
    float tS=0;
    float distancia=0.0;

    GpioDataRegs.GPBCLEAR.bit.GPIO61=1;     //nos aseguramos de que el pin esté bajo
    delay(50);
    GpioDataRegs.GPBSET.bit.GPIO61=1;       //encendemos el pin Trig

    StartCpuTimer0();
    while(CpuTimer0.InterruptCount<1)
           {
               movimiento=reposo;
               movimiento();
           }
    StopCpuTimer0();
    ReloadCpuTimer0();
    CpuTimer0.InterruptCount=0;             //le damos 10uS

    GpioDataRegs.GPBCLEAR.bit.GPIO61=1;     //apagamos el pin Trig

    while(*echo==0)
    {
        movimiento=reposo;
        movimiento();
    }
    while (*echo==1)
    {
        movimiento=reposo;
        movimiento();
    }
                        //controlamos la espera para que el pin echo detecte el
retorno de la señal

    cputime=cputime1-cputime2;
    microS=cputime/150;                     //sacamos el tiempo de respuesta en uS
    tS = microS/1000000;
    distancia =100*(tS*340/2);              //obtenemos la distancia final

    return distancia;
}

void delay (z)
    {
        int N,i;
        N=z*150;
        for(i=0;i<N;i++);
    }

__interrupt void
obstaculo(void)
{
    EALLOW;
    if (echo==0)
    {
        CpuTimer2Regs.TCR.bit.TRB=1;        //reseteamos el timer 2
        CpuTimer2Regs.TCR.bit.TSS=0;        //activamos el timer 2
        cputime1=CpuTimer2Regs.TIM.all;     //tomamos el momento de inicio del timer
2
        echo=1;

    }
```

```
    else
    {
        cputime2=CpuTimer2Regs.TIM.all;     //tomamos el final del timer 2
        CpuTimer2Regs.TCR.bit.TSS=1;        //apagamos el timer 2
        echo=0;
    }

    EDIS;
    PieCtrlRegs.PIEACK.bit.ACK1 = 1;        //habilitamos el registro para volver a
entrar en la interrupción

}
```

## 12. Código en C para selección de GPIO y PWM

```
void Setup_ePWM(void)
{   //pwm1
    // ConfiguraciÃ³n del mÃ³dulo Time-Base
    EPwm1Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm1Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm1Regs.TBCTL.bit.SYNCOSEL = 0;       // SincronizaciÃ³n deshabilitada
    EPwm1Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    //pwm2
    // ConfiguraciÃ³n del mÃ³dulo Time-Base
    EPwm2Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm2Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm2Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm2Regs.TBCTL.bit.SYNCOSEL = 0;       // SincronizaciÃ³n deshabilitada
    EPwm2Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    //pwm3
    EPwm3Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm3Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm3Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm3Regs.TBCTL.bit.SYNCOSEL = 0;       // SincronizaciÃ³n deshabilitada
    EPwm3Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    //pwm6
    EPwm6Regs.TBCTL.bit.CLKDIV = 0;         // Pre-escalador CLKDIV = 1
    EPwm6Regs.TBCTL.bit.HSPCLKDIV = 1;      // Pre-escalador HSPCLKDIV = 2
    EPwm6Regs.TBCTL.bit.CTRMODE = 2;        // Modo up - down
    EPwm6Regs.TBCTL.bit.PRDLD = 0;          // Shadow activado
    EPwm6Regs.TBCTL.bit.SYNCOSEL = 0;       // SincronizaciÃ³n deshabilitada
    EPwm6Regs.TBPRD = 37500;                // Periodo para 1KHz de fPWM

    // ConfiguraciÃ³n del mÃ³dulo Compare-Counter
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm1Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm1Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm2Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm2Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    EPwm3Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm3Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm3Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm3Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    EPwm6Regs.CMPCTL.bit.SHDWAMODE = 0;     // shahow activado para CMPA
    EPwm6Regs.CMPCTL.bit.SHDWBMODE = 0;     // shahow activado para CMPB
    EPwm6Regs.CMPCTL.bit.LOADAMODE = 0;     // shadow de CMPA cargado en TBCTR = 0
    EPwm6Regs.CMPCTL.bit.LOADBMODE = 0;     // shadow de CMPB cargado en TBCTR = 0

    // ConfiguraciÃ³n del mÃ³dulo Action Qualifier
    EPwm1Regs.AQCTLA.all = 0;
    EPwm1Regs.AQCTLA.bit.CAD = 1;           // EPWM1A a low en CMPA down
    EPwm1Regs.AQCTLA.bit.CAU = 2;           // EPWM1A a high en CMPA up
    EPwm1Regs.AQCTLB.all = 0;
    EPwm1Regs.AQCTLB.bit.CBD = 1;           // EPWM1B a low en CMPB down
    EPwm1Regs.AQCTLB.bit.CBU = 2;           // EPWM1B a high CMPB up
```

```
    EPwm2Regs.AQCTLA.all = 0;
    EPwm2Regs.AQCTLA.bit.CAD = 1;           // EPWM2A a low en CMPA down
    EPwm2Regs.AQCTLA.bit.CAU = 2;           // EPWM2A a high en CMPA up
    EPwm2Regs.AQCTLB.all = 0;
    EPwm2Regs.AQCTLB.bit.CBD = 1;           // EPWM2B a low en CMPB down
    EPwm2Regs.AQCTLB.bit.CBU = 2;           // EPWM2B a high CMPB up

    EPwm3Regs.AQCTLA.all = 0;
    EPwm3Regs.AQCTLA.bit.CAD = 1;           // EPWM2A a low en CMPA down
    EPwm3Regs.AQCTLA.bit.CAU = 2;           // EPWM2A a high en CMPA up
    EPwm3Regs.AQCTLB.all = 0;
    EPwm3Regs.AQCTLB.bit.CBD = 1;           // EPWM2B a low en CMPB down
    EPwm3Regs.AQCTLB.bit.CBU = 2;           // EPWM2B a high CMPB up

    EPwm6Regs.AQCTLA.all = 0;
    EPwm6Regs.AQCTLA.bit.CAD = 1;           // EPWM2A a low en CMPA down
    EPwm6Regs.AQCTLA.bit.CAU = 2;           // EPWM2A a high en CMPA up
    EPwm6Regs.AQCTLB.all = 0;
    EPwm6Regs.AQCTLB.bit.CBD = 1;           // EPWM2B a low en CMPB down
    EPwm6Regs.AQCTLB.bit.CBU = 2;           // EPWM2B a high CMPB up

    // ConfiguraciÃ³n del mÃ³dulo Dead Band
    EPwm1Regs.DBCTL.bit.OUT_MODE = 0;       // MÃ³dulo activado
    EPwm2Regs.DBCTL.bit.OUT_MODE = 0;       // MÃ³dulo activado
    EPwm3Regs.DBCTL.bit.OUT_MODE = 0;       // MÃ³dulo activado
    EPwm6Regs.DBCTL.bit.OUT_MODE = 0;       // MÃ³dulo activado



    // ConfiguraciÃ³n del mÃ³dulo PWM-Chopper
    EPwm1Regs.PCCTL.bit.CHPEN = 0;          // MÃ³dulo descativado
    EPwm2Regs.PCCTL.bit.CHPEN = 0;          // MÃ³dulo descativado
    EPwm3Regs.PCCTL.bit.CHPEN = 0;          // MÃ³dulo descativado
    EPwm6Regs.PCCTL.bit.CHPEN = 0;          // MÃ³dulo descativado


    // ConfiguarciÃ³n del mÃ³dulo Trip Zone
    EPwm1Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm1Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm1Regs.TZEINT.all = 0;               // interrupciÃ³n deshabilitada

    EPwm2Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm2Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm2Regs.TZEINT.all = 0;               // interrupciÃ³n deshabilitada

    EPwm3Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm3Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm3Regs.TZEINT.all = 0;               // interrupciÃ³n deshabilitada

    EPwm6Regs.TZCTL.bit.TZA = 3;            // No hacer nada
    EPwm6Regs.TZCTL.bit.TZB = 3;            // No hacer nada
    EPwm6Regs.TZEINT.all = 0;               // interrupciÃ³n deshabilitada


    // ConfiguraciÃ³n del mÃ³dulo Event Trigger
    EPwm1Regs.ETSEL.bit.SOCAEN = 0;         // GeneraciÃ³n de SOCA deshabilitado
    EPwm1Regs.ETSEL.bit.SOCBEN = 0;         // GeneraciÃ³n de SOCB deshabilitado
    EPwm1Regs.ETSEL.bit.INTEN = 0;          // GeneraciÃ³n de seÃ±al de interrupciÃ³n
deshabilitado

    EPwm2Regs.ETSEL.bit.SOCAEN = 0;         // GeneraciÃ³n de SOCA deshabilitado
    EPwm2Regs.ETSEL.bit.SOCBEN = 0;         // GeneraciÃ³n de SOCB deshabilitado
    EPwm2Regs.ETSEL.bit.INTEN = 0;          // GeneraciÃ³n de seÃ±al de interrupciÃ³n
deshabilitado
```

```
    EPwm3Regs.ETSEL.bit.SOCAEN = 0;         // GeneraciÃ³n de SOCA deshabilitado
    EPwm3Regs.ETSEL.bit.SOCBEN = 0;         // GeneraciÃ³n de SOCB deshabilitado
    EPwm3Regs.ETSEL.bit.INTEN = 0;          // GeneraciÃ³n de seÃ±al de interrupciÃ³n
deshabilitado

    EPwm6Regs.ETSEL.bit.SOCAEN = 0;         // GeneraciÃ³n de SOCA deshabilitado
    EPwm6Regs.ETSEL.bit.SOCBEN = 0;         // GeneraciÃ³n de SOCB deshabilitado
    EPwm6Regs.ETSEL.bit.INTEN = 0;          // GeneraciÃ³n de seÃ±al de interrupciÃ³n
deshabilitado

}

void Gpio_select(void)
{
    EALLOW;

    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;     // GPIO0 como EPWM1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;     // GPIO1 como EPWM6A
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;     // GPIO0 como EPWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;     // GPIO1 como EPWM3A


    GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;    // GPIO10 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO12 = 1;     // GPIO10 como salida

    GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;    // GPIO11 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;     // GPIO11 como salida

    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0;    // GPIO11 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1;     // GPIO11 como salida

    GpioCtrlRegs.GPAMUX2.bit.GPIO16 = 0;    // GPIO11 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO16= 1;     // GPIO11 como salida

    GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;  // GPIO13 como E/S
    GpioCtrlRegs.GPBDIR.bit.GPIO61 = 1;  // GPIO13 como salida (trig)

    GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;  // GPIO14 como E/S
    GpioCtrlRegs.GPADIR.bit.GPIO14 = 0;  // GPIO14 como entrada (echo)
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 14;      // Línea 3 de interrupción por
GPIO (XINT3)

    EDIS;
}
```

# 13. Código en C de premapeo

```
void premapeo(void)
{
    float dist;
    int p, q;
    int count=0;
    int prx, pry;
    int bajo=0, subo=0, der=0, izq=0;

    for(p=0 ; count < R*S - 1 ; p++)
    {                                           //INICIO FOR DE FILAS
        if ( p%2 == 0 )
        {                                       //INICIO IF FILAS PARES

            //La columna es par por tanto voy hacia la derecha (++)
            for(q=0 ; q<S; q++)                 //INICIO DE FOR PARA COLUMNAS DE IZQ
A DER
            {
            //En primer lugar colocamos el salto de línea
                if ( q >= S-1 )                 //INCIO DEL IF PARA CAMBIAR DE LÍNEA
                {
                    //giro 90 grados a la derecha
                    giro_der();
                    //
                    dist=medirdistancia(&echo);
                    if ( dist > d_lim )
                    {
                        M[p+1][q] = 0;
                        count++;

                        //avanzo
                        avanzo();
                        //
                        //giro a la derecha 90 grados
                        giro_der();
                        //
                    }
                    else
                    {
                        M[p+1][q] = 1;
                        prx=q;
                        pry=p;

                        //giro a la derecha 90 grados
                        giro_der();
                        //

                        dist=medirdistancia(&echo);
                        if (dist > d_lim)
                        {
                            q++;
                            der=1;
                            bajo=0;
                            subo=0;
                            izq=0;

                            M[p][q] = 0;

                            //avanzo
                            avanzo();
                        //
                        }
                        else
```

```
{
    //giro a la derecha 90 grados
    giro_der();
    //
    dist=medirdistancia(&echo);
    if(dist > d_lim)
            {
                p--;
                subo=1;
                bajo=0;
                der=0;
                izq=0;

                M[p][q] = 0;
                //avanzo
                avanzo();
                //
            }
}

while ((pry + 1) != p)
{
    if ( subo == 1 ) //acabo de bajar
    {
        //giro 90 grados a la derecha
        giro_der();
        //
        dist=medirdistancia(&echo);
        if(dist > d_lim)
        {
            q++;
            der=1;
            bajo=0;
            subo=0;
            izq=0;

            M[p][q] = 0;
            //avanzo
            avanzo();
            //
        }
        else
        {
            //giro 90 grados a la izquierda
            giro_izq();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p--;
                subo=1;
                bajo=0;
                der=0;
                izq=0;

                M[p][q] = 0;

            //avanzo
            avanzo();
            //
            }
        }
    }
    else if( der == 1 )
    {
        //giro 90 grados a la derecha
```

```
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p++;
                bajo=1;
                subo=0;
                izq=0;
                der=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
                //
            }
            else
            {
                //giro 90 grados a la izquierda
                giro_izq();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    q++;
                    der=1;
                    bajo=0;
                    subo=0;
                    izq=0;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                //
                }
                else
                {
                    //giro 90 grados a la izquierda
                    giro_izq();
                    //
                    dist=medirdistancia(&echo);
                    if(dist > d_lim)
                    {
                        p--;
                        subo=1;
                        bajo=0;
                        der=0;
                        izq=0;

                        M[p][q] = 0;

                        //avanzo
                        avanzo();
                //
                    }
                }
            }
        }
        else if( bajo == 1 )
        {
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p++;
```

```
                        bajo=1;
                        subo=0;
                        izq=0;
                        der=0;

                        M[p][q] = 0;

                        //avanzo
                        avanzo();
                        //
                    }
                    else
                    {
                        //giro 90 grados a la izquierda
                        giro_izq();
                        //
                        dist=medirdistancia(&echo);
                        if(dist > d_lim)
                        {
                            q++;
                            der=1;
                            bajo=0;
                            subo=0;
                            izq=0;

                            M[p][q] = 0;

                            //avanzo
                            avanzo();
                        //
                        }
                    }
                }
            }               //FIN DEL WHILE
            count = count + q - prx + 1;
            q--;
            p--;

            //giro de 90 grados a la derecha
            giro_der();
            //
        }                   //FIN DE ELSE IF DE OBSTÁCULO
    }                       //FIN DEL IF PARA SALTO DE LÍNEA

        //ejecución normal sin obstáculos

    dist=medirdistancia(&echo);
    if ( dist > d_lim)
    {
        M[p][q+1] = 0;
        count++;

        //avanzo
        avanzo();
        //
    }

        //ejecución con obstáculos
    else
    {                                   //INICIO DE ELSE IF DE OBSTÁCULO
            M[p][q+1] = 1;
            prx=q;
            pry=p;

            //giro 90 grados a la izquierda
            giro_izq();
```

```
    //

if ( dist > d_lim ) //Puedo bajar?
{                   //INICIO DEL IF PARA SORTEAR OBSTÁCULO
    p--;
    subo=1;

    //avanzo
    avanzo();
    //

    while (pry != p)
    {                           //INICIO DEL WHILE
        if ( subo == 1 ) //acabo de bajar
        {
            //giro 90 grados a la derecha
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                q++;
                der=1;
                bajo=0;
                subo=0;
                izq=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
                //
            }
            else
            {
                //giro 90 grados a la izquierda
                giro_izq();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    p--;
                    subo=1;
                    bajo=0;
                    der=0;
                    izq=0;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                //
                }
            }
        }
        else if( der == 1 )
        {
            //giro 90 grados a la derecha
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p++;
                bajo=1;
                subo=0;
```

```
            izq=0;
            der=0;

            M[p][q] = 0;

            //avanzo
            avanzo();
            //
        }
        else
        {
            //giro 90 grados a la izquierda
            giro_izq();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                q++;
                der=1;
                bajo=0;
                subo=0;
                izq=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
            //
            }
            else
            {
                //giro 90 grados a la izquierda
                giro_izq();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    p--;
                    subo=1;
                    bajo=0;
                    der=0;
                    izq=0;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                    //
                }
            }
        }
    }

    else if( bajo == 1 )
    {
        dist=medirdistancia(&echo);
        if(dist > d_lim)
        {
            p++;
            bajo=1;
            subo=0;
            izq=0;
            der=0;

            M[p][q] = 0;
```

```
                                //avanzo
                                avanzo();
                                //
                            }
                            else
                            {
                                //giro 90 grados a la izquierda
                                giro_izq();
                                //
                                dist=medirdistancia(&echo);
                                if(dist > d_lim)
                                {
                                    q++;
                                    der=1;
                                    bajo=0;
                                    subo=0;
                                    izq=0;

                                    M[p][q] = 0;

                                    //avanzo
                                    avanzo();
                                //
                                }
                            }
                        }
                    }           //FIN DE WHILE
                count = count + (q - prx);
                q--;
                }               //FIN DE IF PARA SORTEAR OBSTÁCULO
            }                   //FIN DE ELSE IF DE OBSTÁCULO
        }                       //FIN DE FOR DE COLUMNAS DE IZQ A DER
    }                           //FIN DE IF DE FILAS

    /************************CAMBIO DE
SENTIDO*****************************************/
    /************************CAMBIO DE
SENTIDO*****************************************/
    /************************CAMBIO DE
SENTIDO*****************************************/
    /************************CAMBIO DE
SENTIDO*****************************************/

    else
    {                       //INICIO DEL ELSE PARA COLUMNAS IMPARES
        //La columna es impar por tanto voy hacia la izquierda (--)
        for(q=S-1 ; q>=0; q--)
        {                   //INICIO DEL FOR PARA IR DE DERECHA A IZQUIERDA

            //En primer lugar se comprueba el fin de carrete
            if ( q <= 0 )
            {                           //INICIO DE IF PARA SALTO DE LÍNEA
                //giro 90º a la izquierda
                giro_izq();
                //
                dist=medirdistancia(&echo);
                if ( dist > d_lim )
                {
                    M[p+1][q] = 0;
                    count++;
                    //avanzo
                    avanzo();
                    //
                    //giro a la izquierda 90 grados
                    giro_izq();
                    //
```

```
}
else
{                               //INICIO ELSE IF PARA BAJAR
    M[p+1][q] = 1;
    prx=q;
    pry=p;

    //giro a la izquierda 90 grados
    giro_izq();
    //
    dist=medirdistancia(&echo);
    if (dist > d_lim)
    {
        q--;
        der=0;
        bajo=0;
        subo=0;
        izq=1;

        M[p][q] = 0;

        //avanzo
        avanzo();
        //
    }
    else
    {
        //giro a la izquierda 90 grados
        giro_izq();
        //
        dist=medirdistancia(&echo);
        if(dist > d_lim)
                {
                    p--;
                    subo=1;
                    bajo=0;
                    der=0;
                    izq=0;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                    //
                }
    }
    while ((pry + 1) != p)
    {                           //INICIO DE WHILE
        if ( subo == 1 ) //acabo de bajar
        {
            //giro 90 grados a la izquierda
            giro_izq();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                q--;
                der=0;
                bajo=0;
                subo=0;
                izq=1;

                M[p][q] = 0;

                //avanzo
                avanzo();
```

```
            //

        }
        else
        {
            //giro 90 grados a la derecha
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p--;
                subo=1;
                bajo=0;
                der=0;
                izq=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
            //
            }
        }
    }
    else if( izq == 1 )
    {
        //giro 90 grados a la izquierda
        giro_izq();
        //
        dist=medirdistancia(&echo);
        if(dist > d_lim)
        {
            p++;
            bajo=1;
            subo=0;
            izq=0;
            der=0;

            M[p][q] = 0;

            //avanzo
            avanzo();
            //
        }
        else
        {
            //giro 90 grados a la derecha
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                q--;
                der=0;
                bajo=0;
                subo=0;
                izq=1;

                M[p][q] = 0;

                //avanzo
                avanzo();
            //
            }
            else
```

```
                {
                    //giro 90 grados a la derecha
                    giro_der();
                    //
                    dist=medirdistancia(&echo);
                    if(dist > d_lim)
                    {
                        p--;
                        subo=1;
                        bajo=0;
                        der=0;
                        izq=0;

                        M[p][q] = 0;

                        //avanzo
                        avanzo();
                //
                    }
                }
            }
        }

        else if( bajo == 1 )
        {
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                p++;
                bajo=1;
                subo=0;
                izq=0;
                der=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
                //
            }
            else
            {
                //giro 90 grados a la derecha
                giro_der();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    q--;
                    der=0;
                    bajo=0;
                    subo=0;
                    izq=1;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                //
                }
            }
        }
    }                   //FIN DE WHILE
    count = count + prx - q + 1;
    q++;
    p--;
```

```
        //giro de 90 grados a la izquierda
        giro_izq();
        //
    }                       //FIN DE ELSE IF PARA BAJAR
}                           //FIN DE IF PARA SALTO DE LÍNEA

dist=medirdistancia(&echo);
if ( dist > d_lim )
{
    M[p][q-1] = 0;
    count++;

    //avanzo
    avanzo();
    //
}
else
{               //INICIO ELSE DE OBSTÁCULO

        M[p][q-1] = 1;
        prx=q;
        pry=p;
    //giro 90 grados a la derecha
        giro_der();
    //
    dist=medirdistancia(&echo);
    if ( dist > d_lim ) //Puedo bajar?
    {                           //INICIO IF PARA SORTEAR OBSTÁCULO
        p--;
        subo=1;

        //avanzo
        avanzo();
    //

        while (pry != p)
        {                           //INICIO WHILE
            if ( subo == 1 ) //acabo de bajar
            {
                //giro 90 grados a la izquierda
                giro_izq();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    q--;
                    der=0;
                    bajo=0;
                    subo=0;
                    izq=1;

                    M[p][q] = 0;

                    //avanzo
                    avanzo();
                    //
                }
                else
                {
                    //giro 90 grados a la derecha
                    giro_der();
                    //
                    dist=medirdistancia(&echo);
                    if(dist > d_lim)
                    {
```

```
                p--;
                subo=1;
                bajo=0;
                der=0;
                izq=0;

                M[p][q] = 0;

                //avanzo
                avanzo();
            //
            }
        }
    }
    else if( izq == 1 )
    {
        //giro 90 grados a la izquierda
        giro_izq();
        //
        dist=medirdistancia(&echo);
        if(dist > d_lim)
        {
            p++;
            bajo=1;
            subo=0;
            izq=0;
            der=0;

            M[p][q] = 0;

            //avanzo
            avanzo();
            //
        }
        else
        {
            //giro 90 grados a la derecha
            giro_der();
            //
            dist=medirdistancia(&echo);
            if(dist > d_lim)
            {
                q--;
                der=0;
                bajo=0;
                subo=0;
                izq=1;

                M[p][q] = 0;

                //avanzo
                avanzo();
            //
            }
            else
            {
                //giro 90 grados a la derecha
                giro_der();
                //
                dist=medirdistancia(&echo);
                if(dist > d_lim)
                {
                    p--;
                    subo=1;
                    bajo=0;
                    der=0;
```

```
                                izq=0;

                                M[p][q] = 0;

                                //avanzo
                                avanzo();
                            //
                                }
                            }
                        }
                    }

                    else if( bajo == 1 )
                    {
                        dist=medirdistancia(&echo);
                        if(dist > d_lim)
                        {
                            p++;
                            bajo=1;
                            subo=0;
                            izq=0;
                            der=0;

                            M[p][q] = 0;

                            //avanzo
                            avanzo();
                            //
                        }
                        else
                        {
                            //giro 90 grados a la derecha
                            giro_der();
                            //
                            dist=medirdistancia(&echo);
                            if(dist > d_lim)
                            {
                                q--;
                                der=0;
                                bajo=0;
                                subo=0;
                                izq=1;

                                M[p][q] = 0;

                                //avanzo
                            avanzo();
                            //
                            }
                        }
                    }
                }           //FIN DE WHILE
                count = count + (prx - q);
                q++;
            }               //FIN DE IF PARA SORTEAR OBSTÁCULO
        }                   //FIN DE ELSE IF PARA SORTEAR OBSTÁCULO
    }                       //FIN DE FOR PARA IR DE DERECHA A IZQUIERDA
}                           //FIN DE ELSE PARA FILAS IMPARES
                            //FIN DE FOR DE FILAS

for(p=0;p<R;p++)
{
    for(q=0;q<S;q++)
        if(M[p][q]==2)
        {
            M[p][q]=1;}}}
```